1. Arquitectura del sistema; arquitectura del microprocesador.

1.a. EL COMPUTADOR

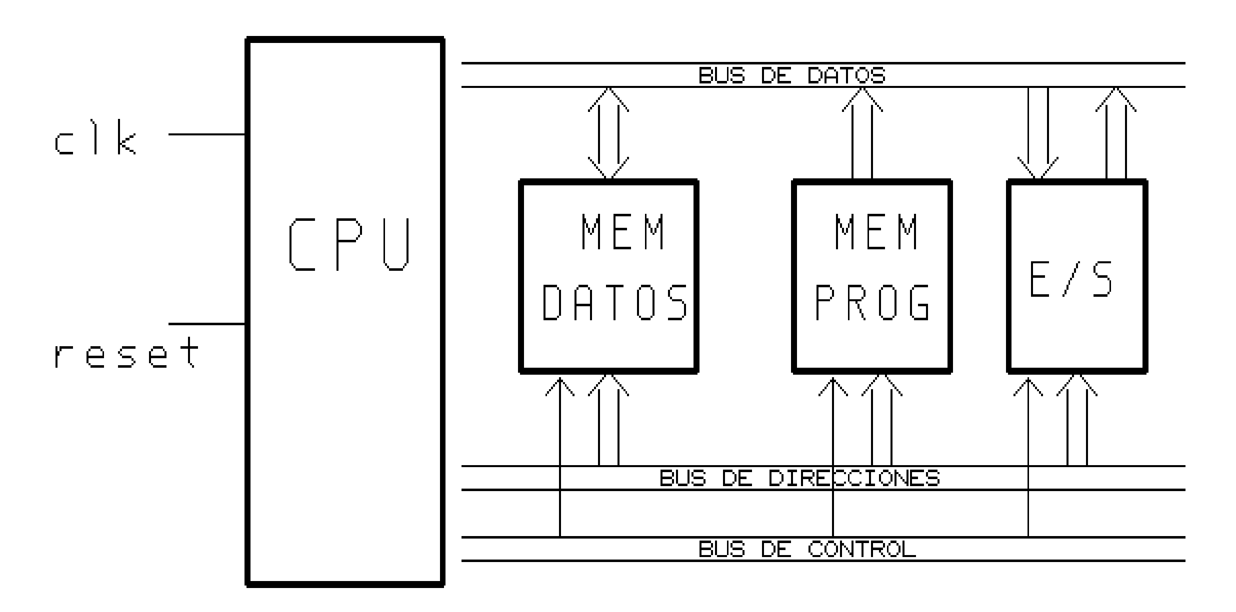


Fig. 1

La Fig. 1 muestra un diagrama de bloques básico de un computador. Entendemos por tal, *una máquina de propósito general, capaz de interpretar y ejecutar una serie de instrucciones.*

Describiremos a continuación los distintos elementos que forman un computador.

En primer lugar, ubicamos la unidad central de proceso, para la cual generalmente se utiliza la sigla inglesa **CPU**. Esta unidad es "el corazón" del computador, y su tarea es *ir tomando ordenadamente las instrucciones del programa almacenado en la memoria de programa e ir interpretando su significado, para luego ejecutarlas.* Existen dos señales que siempre llegan a la CPU:

RELOJ = Dado que se trata de un sistema secuencial muy complejo siempre es modo reloj, y existe una única señal de reloj para todo el sistema.

RESET = Dado que nuestro sistema secuencial tiene una gran cantidad de estados posibles, es necesario poder forzarlo a un estado inicial conocido (por ejemplo, inmediatamente después de ponerlo en funcionamiento): para ello se utiliza la línea de RESET.

La memoria de programa consiste en *el lugar físico donde se almacenan la serie de instrucciones que el computador deberá*

ejecutar, en tanto que la memoria de datos es el lugar donde *se almacenan los datos que utilizan (o generan) los programas que corre la computadora.* La separación entre las memorias de programa y de datos puede ser física (tengo espacio de direccionamiento separado para programa y datos) o simplemente lógica.

Puede verse también en la Fig. 1 un bloque marcado E/S, que incluye a *todos aquellos dispositivos por intermedio de los cuales el computador intercambia información con el medio que lo rodea.*

Un concepto muy importante - y también muy sencillo - es el de bus: se trata de *un conjunto de conductores eléctricos que funcionalmente forman una unidad*. Típicamente, todo computador tiene los tres buses indicados en la Fig. 1 , a saber:

El bus de datos, que transporta datos de unos elementos del sistema a otros, por ejemplo, entre la memoria de programa y la CPU, o entre la CPU y un periférico.

El bus de direcciones lleva las direcciones generadas por la CPU, para seleccionar un registro interno del sistema (luego diremos "una posición de memoria o de I/O") sobre el cual operar.

El bus de control lleva diversas señales de sincronización que gobiernan el funcionamiento del sistema.

1.b. EL MICROPROCESADOR

El avance de la microelectrónica hizo posible que cada uno de los bloques de una computadora pudiera fabricarse en un chip.
Se llama **MICROPROCESADOR** a la implementación de una CPU en un solo chip. El primer microprocesador disponible comercialmente fue fabricado en 1971 por Intel Corporation. Se llamaba 4004, era de 4 bits, y era un chip de 16 patas.

Los computadores, y también los microprocesadores se caracterizan por el tamaño de su palabra de datos, es decir, por el número de bits que tiene la unidad de información que manejan. Así, diremos que un microprocesador es de 8 bits, si cuando suma, guarda en memoria, escribe en un periférico, etc., lo hace en unidades de 8 bits. El tamaño de su palabra de datos es siempre igual al número de líneas que posee el bus de datos.

Otro aspecto que caracteriza a un microprocesador es su capacidad de direccionamiento: se trata del tamaño de la memoria que puede direccionar: si el bus de direcciones de un procesador tiene n líneas, entonces tendrá una capacidad de direccionamiento de 2^n palabras.

Un caso particular de microprocesador, es el Z-80: **es un procesador de 8 bits (su tamaño de palabra es de 8 bits) y tiene una capacidad de direccionamiento de 64 Kbytes (= 2^{16}** bytes).

Todo lo dicho sobre el sistema general es perfectamente válido aquí. Simplemente, ahora estamos viendo un ejemplo concreto , y antes se trataba de un caso general, abstracto.

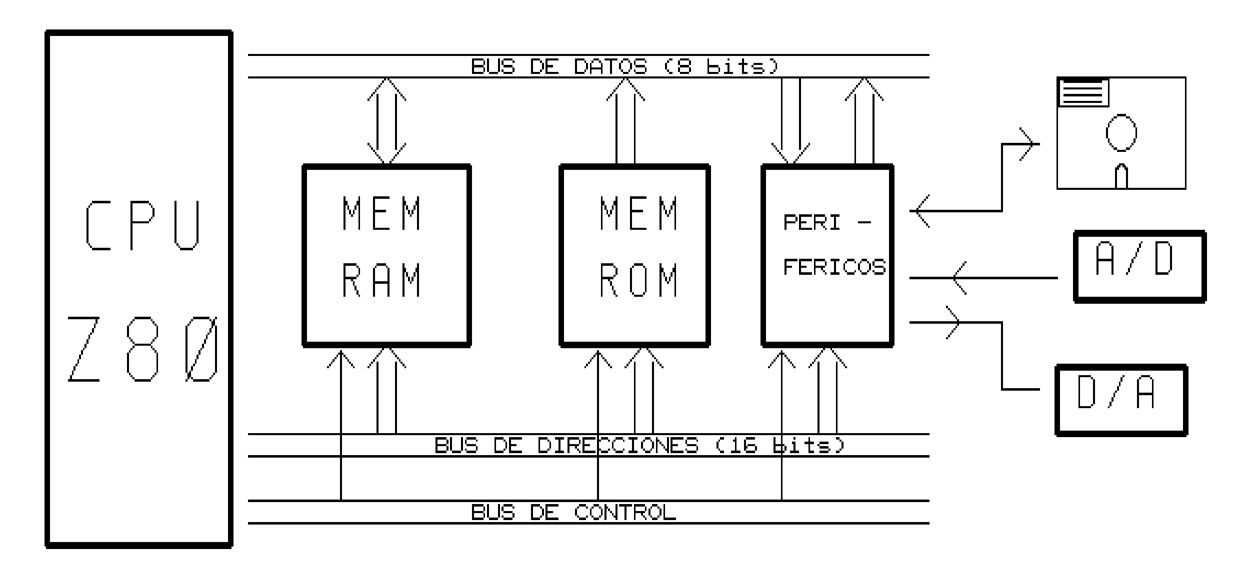


Fig. 2 - Sistema basado en un microprocesador Z80

Los bloques "memoria de programa" y "memoria de datos" ahora están implementados por medio de ROM y RAM.

La memoria ROM (*Read Only Memory*) es memoria de solo lectura, cuyo contenido no se altera a pesar de que se interrumpa la alimentación de energía, y por lo tanto, se utiliza para almacenar los programas que ejecuta el microprocesador, o por lo menos, <u>el primer programa que ejecuta una vez que se pone en marcha el sistema</u>. Quiere decir que, al menos una parte de lo que en la Fig. 1 denominábamos memoria de programa, será ahora implementado como una ROM.

En cuanto a la RAM, esta se utiliza para almacenar temporariamente los datos sobre los que está trabajando el programa, y también se puede utilizar para almacenar <u>programas</u> temporariamente: el único inconveniente es que en caso de falta de energía ellos se pierden. Este problema se soluciona si disponemos - como en la Fig. 2 - de un dispositivo de entrada-salida adecuado para almacenar datos o programas, como por ejemplo, una unidad de disco flexible.

El bloque "E/S" se sustituye ahora por otros, denominados genéricamente "periféricos", que implementan la comunicación del sistema con el mundo exterior. Su complejidad va desde simples interruptores a otros sistemas con microprocesador que preprocesan datos antes de ser entrados al microprocesador principal.

Para entender los sistemas con microprocesadores, es ventajoso ver el sistema completo - microprocesador, ROM, RAM, puertos de E/S - como una colección de registros direccionables. Los registros que residen en el interior del microprocesador se denominan <u>registros internos</u>, y aquellos que existen en ROM, RAM y puertos de E/S son los <u>registros externos.</u>

El conjunto de registros que constituyen un sistema en

particular y las transferencias de datos que sean posibles entre ellos forman la arquitectura del sistema. Los tipos de registro en el microprocesador y las posibles transferencias de datos entre ellos determinan la arquitectura del microprocesador.

Un sistema con microprocesador implementa sus funciones a base de *transferir y transformar* datos entre registros del sistema. Típicamente, las transformaciones de los datos tienen lugar en los registros internos, muchos de los cuales son registros de operación. Los *registros de operación* se diferencian de los de almacenamiento en que contienen los operandos para las operaciones aritméticas y lógicas.

El microprocesador controla y sincroniza las transferencias de datos y las transformaciones de acuerdo con unas instrucciones leídas desde el programa de aplicación residente en la ROM del sistema (o eventualmente en RAM, como ya se mencionó).

1.c. ARQUITECTURA INTERNA DEL Z80

En la Fig. 3 vemos un esquema de la arquitectura interna del Z80. Es preciso destacar que este diagrama no es para nada exhaustivo, y se utiliza aquí simplemente porque da una idea aproximada de la estructura interna del microprocesador.

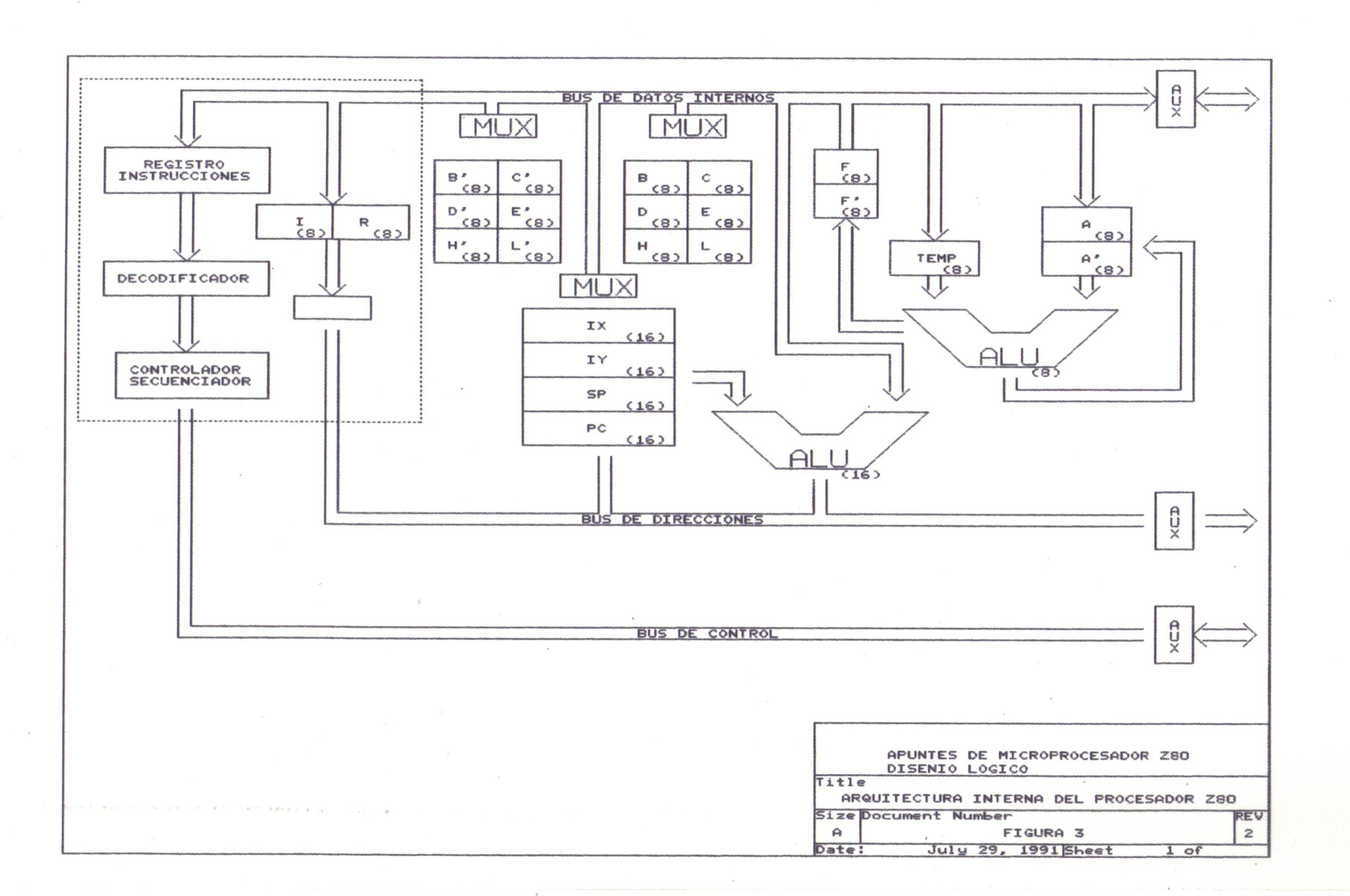
1.c.I. UNIDAD DE CONTROL Y REGISTROS INTERNOS ASOCIADOS

Se trata de la parte recuadrada en la Fig. 3 . Es quien controla y sincroniza la todas las transferencias de datos y sus transformaciones en el sistema con microprocesador, y es el subsistema secuencial clave del microprocesador en sí mismo. Las acciones atribuibles al microprocesador son acciones implementadas por la unidad de control.

La unidad de control utiliza entradas del reloj maestro para derivar señales de los tiempos y de control que regulan las transferencias y transformaciones en el sistema, asociadas con cada instrucción. La unidad de control acepta también, como entrada, señales de control generadas por otros dispositivos del sistema con microprocesador, que alteran el estado del microprocesador (Ver Fig. 4)

La operación básica de un microprocesador se regula mediante la unidad de control, es cíclica y consiste en la búsqueda y posterior ejecución secuencial de instrucciones. Cada ciclo de ejecución de instrucción tiene dos estados básicos: el estado de búsqueda y el estado de ejecución. El primero, transfiere una instrucción desde memoria al microprocesador y el segundo, ejecuta la instrucción.

Para realizar esas tareas, se utilizan los registros que aparecen en la Fig. 3 .Existe un registro de instrucción, cuya finalidad es almacenar la instrucción que se ha buscado hasta que esta sea decodificada y ejecutada. Se tiene también el decodificador, que envía señales al controlador-secuenciador y determina la ejecución de la instrucción dentro y fuera del microprocesador.


BUS DE DATOS INTERNOS
REGISTRO INSTRUCCIONES
DECODIFICADOR
CONTROLADOR SECUENCIADOR
I (8)
R (8)
MUX
MUX
MUX
B' (8)
C' (8)
D' (8)
E' (8)
H' (8)
L' (8)
B (8)
C (8)
D (8)
E (8)
H (8)
L (8)
IX (16)
IY (16)
SP (16)
PC (16)
F (8)
F' (8)
TEMP (8)
A (8)
A' (8)
ALU (8)
ALU (16)
AUX
AUX
AUX
BUS DE DIRECCIONES
BUS DE CONTROL
APUNTES DE MICROPROCESADOR Z80
DISENIO LOGICO
Title
ARQUITECTURA INTERNA DEL PROCESADOR Z80
Size
A
Document Number
FIGURA 3
REV
2
Date: July 29, 1991
Sheet 1 of

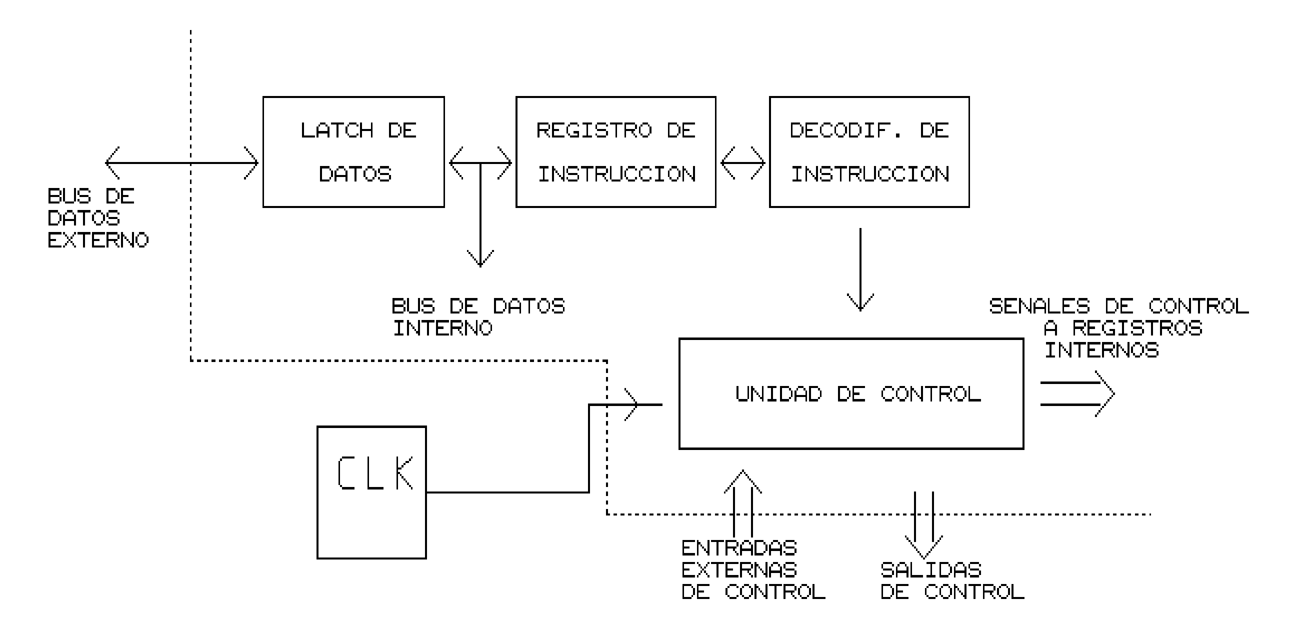


Fig. 4

Para buscar la instrucción que debe ser ejecutada a continuación, la unidad de control mantiene un registro de propósito especial, el contador de programa (PC). Se trata de un registro de 16 bits que contiene la posición de la instrucción que actualmente está siendo buscada de memoria. Al completarse la ejecución de una instrucción, el contador de programa contiene la dirección de la primera palabra de la siguiente instrucción a ejecutar.

En el caso del Z80, las instrucciones tienen una longitud entre 1 y 4 bytes. El programador puede cargar o guardar el contenido de PC, pero no puede realizar operaciones aritméticas con él.

A los efectos de colocar la dirección contenida en el PC en el bus de direcciones, se copia el contenido del mismo en el registro auxiliar del bus de direcciones. La salida de este registro son las patas de dirección del microprocesador. La unidad de control genera entonces una señal de lectura de memoria (a través de las líneas que correspondan del bus de control) que transfiere el dato de la posición de memoria direccionada al microprocesador.

El dato se transfiere a través del bus de datos del sistema, el registro auxiliar del bus de datos y el bus de datos interno del Z80, hasta alcanzar el registro de instrucciones ya mencionado. El primer byte de una instrucción (y en algunos casos, los dos primeros bytes) constituyen el código de operación (también llamado OPCODE) para dicha instrucción. Una vez decodificado, éste indica a la unidad de control las operaciones requeridas para ejecutar la instrucción. El código de operación direcciona una posición inicial en una ROM de control o PLA dentro del microprocesador , donde hay una secuencia de instrucciones muy elementales, llamadas microinstrucciones. Cada instrucción en el conjunto fijo de instrucciones de un microprocesador se implementa por la unidad de

control, secuenciando un conjunto de micro-operaciones asociadas con una instrucción en particular. Para los microprocesadores en un solo chip, la microinstrucción, y por lo tanto el conjunto de instrucciones del microprocesador, está fijada en la fabricación.

Las instrucciones del microprocesador a menudo requieren más información de la proporcionada en una única palabra de memoria. Los bytes sucesivos representan una dirección o una constante de datos. La existencia o no de estos bytes, y su significado, es información contenida en el código de operación de cada instrucción.

1.c.II. LA UNIDAD ARITMETICA Y LOGICA

La ALU realiza las operaciones aritméticas y lógicas, que constituyen las transformaciones básicas de datos implementadas en un microprocesador. En una de sus entradas esta el ACUMULADOR, en nuestro caso el registro A, que es un registro especial, ya que es el único que se puede utilizar para realizar ciertas operaciones. En particular, para las operaciones aritméticas y lógicas de dos operandos, uno de ellos estará obligatoriamente en el acumulador. Además, el resultado de la operación generalmente queda en el acumulador. En la otra entrada tengo un registro temporal o BUFFER, que almacena temporariamente el otro operando que utilizará la ALU.

El registro de estado, que contiene las banderas , se llama **F** en el Z80. Su función es indicar las situaciones excepcionales que se dan en el interior del microprocesador, por ejemplo: habrá una bandera que indicará si el resultado de la última operación fue cero.

El estado de las banderas se utiliza para controlar el flujo del programa, por ejemplo, ejecutar una determinada rutina si el resultado de la última suma fue cero.

El formato del registro de estado es el siguiente:

7	6	5	4	3	2	1	0
S	Z	----	H	----	P/V	N	C

S - Signo
Z - Cero
H - Acarreo del tercer al cuarto bit para operaciones BCD.
P/V - Doble propósito: paridad/ overflow
N - Bandera de resta, para resta BCD.
C - Carry.

Vemos en la Fig. 3 que tanto el acumulador como el registro de estado aparece duplicado en el Z80: tenemos los acumuladores A y A’, y los registros de estado F y F’.
El programador deberá elegir si trabaja con la pareja A y F o A’y F’, ya que solo puede utilizarse un juego a la vez. En caso de querer cambiar de registros, existe una instrucción para inter-

cambiar los valores de los registros A y F con los de A’ y F’.

1.c.III. REGISTROS DE PROPOSITO GENERAL

Además de los 2 acumuladores, el Z80 posee 12 registros de uso general, de 8 bits: B, C, D, E, H y L por un lado y B’, C’, D’, E’, H’ y L’ por otro. Estos conjuntos forman 2 bancos, de los cuales uno solo puede estar activo en un momento dado, y también existe una instrucción para seleccionar un banco o el otro. Por lo dicho anteriormente, se puede tener activos los siguientes conjuntos de registros:

A y F ---- B, C, D, E, H y L

A’ y F’ ---- B, C, D, E, H y L

A y F ---- B’, C’, D’, E’, H’ y L’

A’ y F’ ---- B’, C’, D’, E’, H’ y L’

Esta peculiaridad es muy interesante pues permite efectuar muy rápidamente el "cambio de contexto", como veremos al estudiar el tema de interrupciones.

Los registros de uso general pueden ser vistos como memoria muy rápida, y por lo tanto privilegiada, de la que dispone el microprocesador. Puede realizarse una transferencia de datos directamente entre cualquiera de ellos y memoria, y además, pueden utilizarse como segundo operando en las operaciones lógicas y aritméticas.

El multiplexor que aparece sobre los bancos de registros se utiliza para seleccionar cuál de ellos se conecta al bus interno del Z80.

Otra característica de los registros B, C, D, E, H y L es que, agrupados por pares B,C; D,E y H,L, se comportan como registros de 16 bits. En muchas operaciones, el contenido de estos registros representa una dirección de memoria. Por ejemplo, el par H,L deriva su nombre de High memory address y Low memory address. Por esta razón, diremos que estos tres pares pueden trabajar como punteros a memoria: entendemos por puntero a un registro cuyo contenido es una dirección de memoria.

Estos pares de registros también pueden ser utilizados en aritmética de doble precisión: pueden efectuarse sumas, restas, incrementos y decrementos con operandos de 16 bits. A esos efectos tenemos otra ALU, como se indica en el esquema.

1.c.IV. REGISTROS DE DIRECCIONES

Son registros destinados específicamente al almacenamiento de direcciones, por ello llamados también punteros. Están conectados con el bus de direcciones, a quien manejan. Para inicializarlos, solo puedo utilizar el bus de datos, que es de 8 bits: debo, por lo

tanto, hacerlo en dos etapas.
En el caso del Z80, tenemos cuatro registros de direcciones: el PC ya explicado, el SP (Puntero de Stack), IX e IY.

1.c.V. EL STACK Y EL PUNTERO DE STACK.

El stack (se traduciría como "la pila") es una estructura de datos LIFO (last in, first out), es decir, el último dato que entró en ella es el primero que saldrá. En la Fig. 5 se esquematiza un stack: se introdujeron los elementos 1, 2, 3, 4, en ese orden. Al retirarlos, saldrán en el orden 4, 3, 2, 1. Esto hace que sea una estructura **cronológica**: lo primero que entró siempre ocupa el fondo del stack, y sobre él se han ido "apilando" los nuevos datos. Puede visualizarse pensando en una pila de latas de conserva en un supermercado: Las dos operaciones que se realizan sobre ella serán: poner una lata encima de todas las demás, para que a partir de ahora sea parte de la pila, o quitar una lata de la pila: obviamente, no intentaré quitar la de más abajo!!!. La operación de poner un dato en el stack, será PUSH (empujar) y la de extraer un dato, POP.

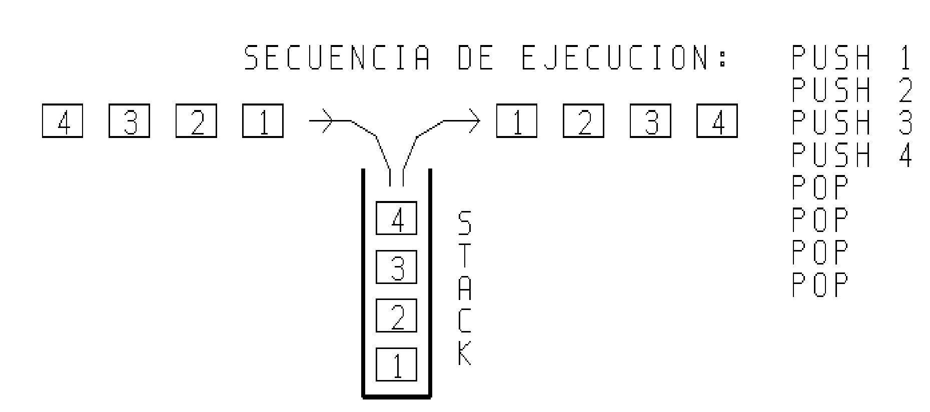


Fig. 5

Prácticamente todo sistema informático tiene un Stack, que se utiliza esencialmente para tres tipos de actividades:

- **Subrutinas**: Cuando desde un programa principal se llama a una subrutina, se guarda en el stack el contenido del PC antes de saltar a la subrutina, y la instrucción RET se encarga de sacar del stack el valor del PC y restituirlo.

-**Interrupciones**: que estudiaremos detalladamente más adelante

-**Almacenamiento temporal de datos**: muchas veces se requiere "desocupar" un registro para utilizarlo con otra finalidad: en lugar de realizar un acceso a memoria común, puedo guardar el valor de ese registro en el Stack.

Existen dos formas de implementar un stack:

a) **Interno:** Se reservan un conjunto de registros internos del microprocesador que están dedicados a implementar el Stack: este resulta muy rápido pero su tamaño es fijo. Se trata de un stack en soporte físico.

b) **Externo:** En el microprocesador solo existe un registro, el puntero de Stack, que almacena la dirección de memoria donde está

el elemento superior de la pila (el último que se introdujo). Se provee pues, solo del soporte lógico del stack. Esta es la solución más usual, y la empleada en el Z80.

En el caso del Z80, el Stack trabaja con datos de 16 bits y además, crece hacia abajo. Esto significa que una operación PUSH, que introduce un dato en el stack, implica un decremento de 2 del SP, y el POP un incremento en 2.

Ejemplo

Consideremos que, en un cierto instante, el contenido de los registros del procesador es el indicado en la tabla 1.

A	03	1F	**F**
B	55	38	**C**
D	AA	BC	**E**
H	C1	37	**L**

1 - Estado inicial del banco de registros

Consideremos también que inicialmente el valor de SP es:

SP = 24F8

Se realizan las siguientes operaciones:

```
PUSH AF
  (queda SP = 24F6)
PUSH DE
  (queda SP = 24F4)
POP BC
  (queda SP = 24F6)
PUSH HL
  (queda SP = 24F4)
```

Dirección	Luego de las 3 primeras instrucciones. SP = 24F6	Dirección	Estado final SP = 24F4
	42		42
	31		31
24F8	A0	24F8	A0
7	03	7	03
6	1F	6	1F
5	AA	5	C1
4	BC	4	37
3		3	

2 - Evolución del Stack.

En el cuadro 2, se observa primero, el estado de la memoria luego de realizadas las tres primeras operaciones. Observar que el hecho de realizar un POP no significa ninguna alteración en la memoria: "quitar" del stack es solamente alterar el puntero y copiar el contenido de ese par de bytes en los registros destino.

La situación mostrada por la columna de la derecha del cuadro 2 muestra el estado final del stack.

En el cuadro 3 vemos cómo quedan los registros luego de realizar estas operaciones: el efecto combinado de la segunda y tercera consiste en copiar en BC el contenido de los registros DE. Dado que sólo existe una operación POP, solo se altera una pareja de registros.

1.c.VI. REGISTROS INDICES

El Z80 tiene dos registros índices: el IX y el IY, que se utilizan en los modos de direccionamiento indexados. Permiten trabajar con estructuras de datos complejas, como por ejemplo, vectores. Para construir la dirección del dato, en el caso de direccionamiento indexado, se toma un byte de "desplazamiento" que está contenido en la propia instrucción, se suma su contenido automáticamente al registro índice que se esté usando, y se accede al dato que está en la dirección de memoria que resulte de dicha operación.

A	03	1F	**F**
B	AA	BC	**C**
D	AA	BC	**E**
H	C1	37	**L**

3 - Estado final de los registros.

Volveremos sobre el tema al estudiar los modos de direccionamiento.

1.c.VII. REGISTRO DE INTERRUPCIÓN Y DE REFRESCO DE MEMORIA

El registro I se utiliza para saber dónde está la rutina que se debe ejecutar cuando el procesador recibe una interrupción, y lo estudiaremos en detalle cuando veamos interrupciones. El registro R se utiliza para refresco de memoria dinámica. Esta memoria tiene como característica que si no se lee su contenido periódicamente (típicamente cada 2 mseg), éste se pierde. El fabricante del Z80 brinda un mecanismo para realizar esa lectura periódica sin necesidad de agregar hardware adicional. Fue una novedad en su momento, y no es usual. Generalmente se utilizan otros integrados que se encargan de esta tarea.

2. Modos de direccionamiento

Cada subsistema en un sistema basado en un microprocesador, - la memoria principal, el microprocesador y los dispositivos de entrada-salida - puede considerarse en términos de los registros que contiene.

Un sistema de este tipo, implementa sus funciones mediante una secuencia de transferencias de datos entre los registros ubicados en memoria principal, en el microprocesador y en dispositivos de E/S, y también mediante transformaciones de datos que ocurren principalmente en los registros internos del microprocesador.

Los tipos de transferencias y transformaciones posibles están especificados en el conjunto de instrucciones (frecuentemente se lo denomina "set de instrucciones" en la literatura). Pero el solo conocimiento del conjunto de instrucciones no permite determinar la potencia del microprocesador. Es necesario conocer también los modos de direccionamiento que es capaz de emplear el microprocesador, es decir, las diferentes maneras que tiene el microprocesador de hacer referencia a los operandos de cada una de las instrucciones que puede ejecutar.

Dicho en forma más rigurosa, *se llama direccionamiento, a la especificación, dentro de una instrucción, de la posición del operando*[1] *sobre el que actuará la misma.*

Es bueno advertir en éste momento, que el estudio de los modos de direccionamiento y del conjunto de instrucciones están íntimamente ligados, por lo cual al estudiar el primero de ellos, deben irse adelantando un cierto número de conceptos referentes al segundo tema. Esta situación puede generar cierta inseguridad en el lector, pero debe tenerse en cuenta que en el capítulo siguiente se tratará en detalle el conjunto de instrucciones.

El Z80 posee un variado repertorio de instrucciones, de diferente grado de complejidad y diferente longitud. También existen diferencias en la forma en que el microprocesador direcciona los operandos. Algunas instrucciones no tienen operandos. Otras, requieren un operando de un registro interno y otro operando que puede provenir de otro registro interno o de la memoria externa. Este segundo operando, puede ser especificado de varias maneras distintas. Por ejemplo, la instrucción ADD suma dos operandos de 8 bits. Uno de los operandos será el registro A, mientras que el segundo puede ser otro registro de la CPU (Direccionamiento a registro), un valor inmediato que esté contenido en la propia instrucción ADD (direccionamiento inmediato), una posición de memoria apuntada por el contenido del par de registros HL (direccionamiento indirecto por registro), o una posición de

[1] - Más en general, puede ser algunas veces también, la posición de memoria donde se almacena el resultado de una operación, o la dirección de la siguiente instrucción a ejecutar.

memoria cuya dirección se calcula sumando un desplazamiento de 8 bits contenido en la instrucción al contenido de un registro índice (direccionamiento indexado).

Lo que resta del presente capítulo está dedicado a describir los distintos modos de direccionamiento del Z80.

2.a. DIRECCIONAMIENTO IMPLICITO

El código de operación de la instrucción es fijo. No existen campos variables dentro del OPCODE, y la instrucción siempre realiza exactamente la misma función.

Ejemplos:

Mnemónico	Opcode[2]	función
CPL	2F	(complementar acumulador)
EX DE, HL	EB	(DE <--> HL)

En el primer ejemplo, se toma el contenido del acumulador A, se forma el complemento a 1 del mismo (cambiando todos los ceros por unos y todos los unos por ceros) y se guarda el resultado en A. Vemos como la "fuente" y el "destino" de la operación son fijos, y no puede utilizarse otro registro.

En el segundo ejemplo, se trata de intercambiar los contenidos de dos pares de registros.

Todas las instrucciones que veremos en el próximo capítulo bajo "Instrucciones Aritméticas de propósito general y control de CPU" así como las de "Intercambio, transferencia de bloques y búsqueda" utilizan este modo de direccionamiento.

2.b. DIRECCIONAMIENTO INMEDIATO

En este caso, el segundo o tercer byte del OPCODE es, él mismo, el operando. Se utiliza cuando se quiere realizar una operación aritmética o lógica con una constante.

Ejemplo:

Mnemónico	Opcode	función
OR 7FH	F6 7F (dato inmediato)	(A <-- A OR 7F H)

Aquí se realiza el OR del acumulador con la constante 7FH, es decir, con el número binario 01111111.

En general, este modo de direccionamiento se utiliza para las

---

[2] - El código de operación se expresará en hexadecimal.

instrucciones aritméticas y lógicas de 8 bits.

2.c. DIRECCIONAMIENTO INMEDIATO EXTENDIDO

Es igual al anterior, pero para los casos en que se requiere una constante de 16 bits. Se utiliza solamente en algunas instrucciones de transferencia de datos de 16 bits.

Ejemplos:

Mnemónico	Opcode		función
LD BC, aabb	01		BC <-- aabb
	bb	(byte bajo de la cte.)	
	aa	(byte alto de la cte.)	
LD IY, aabb	FD		IY <-- aabb
	21		
	bb		
	aa		

Nótese que independientemente del opcode (que en el primer ejemplo ocupa un solo byte, y en el segundo ocupa dos), los dos últimos bytes de la instrucción contienen el dato inmediato. En estos ejemplos aparece por primera vez un concepto importante: *siempre que el Z80 necesite almacenar datos de más de un byte de extensión en memoria, guardará el byte menos significativo del dato en la posición de memoria cuya dirección es más baja*. Si se revisa el ejemplo propuesto al explicar el funcionamiento del stack, se verá que allí también se verifica esta afirmación.

2.d. DIRECCIONAMIENTO POR REGISTRO

En este caso, uno o más de los registros de la CPU es direccionado por la instrucción, que, por ese motivo contiene uno o varios campos que especifican el o los registros que se deben utilizar.

Ejemplos:

Mnemónico	Opcode (bits)	función
XOR R	1 0 1 0 1 r r r	or exclusivo entre A y el registro según valor de r
SRL R	1 1 0 0 1 0 1 1 0 0 1 1 1 r r r	Desplaz. lógico a la derecha del reg. R

R está codificado mediante los tres bits rrr, como sigue:

A: 111	C: 001	E: 011	L: 101
B: 000	D: 010	H: 100	

Nótese que todas las combinaciones de bits son permitidas

excepto 110. Este valor se utiliza para indicar que se está utilizando otro modo de direccionamiento.

Los grupos de instrucciones que utilizan este modo de direccionamiento son las aritméticas y lógicas de 8 bits, las aritméticas de 16 bits, las de rotación y desplazamiento, y las de bit set, reset y test.

2.e. DIRECCIONAMIENTO INDIRECTO POR REGISTRO

Consiste en utilizar los pares de registros BC, DE, y HL como punteros a memoria. Este modo de direccionamiento es práctico para referenciar datos que están secuencialmente en memoria, no así los que se encuentran desordenados. Esto se debe a que en éste último caso, por cada acceso a memoria debe cargarse previamente el registro puntero con la dirección del nuevo byte de datos. En cambio, si los datos se encuentran en posiciones contiguas, se accede a éstos en forma más sencilla, utilizando instrucciones que incrementan o decrementan automáticamente los punteros.

Ejemplo:

supongamos la situación:
HL = 1000H
Se ejecuta
la instrucción
INC (HL)

1000:	34
1001:	78

Posteriormente, en la posición 1000H de memoria, estará el valor 35.

Este modo de direccionamiento se utiliza principalmente en instrucciones de transferencia de 8 bits, instrucciones lógicas y aritméticas de 8 bits, e instrucciones de rotación y desplazamiento.

2.f. DIRECCIONAMIENTO DIRECTO O EXTENDIDO.

Las instrucciones que utilizan este modo de direccionamiento llevan en la propia instrucción la dirección del dato.

Ejemplos:

Mnemónico	Opcode	función
JP nn	C3 byte bajo de nn byte alto de nn	(PC <-- nn)
LD IX,(nn)	DD 2A byte bajo de nn byte alto de nn	(IX <-- (nn))

Este modo de direccionamiento se utiliza principalmente para

las instrucciones de transferencia de datos de 8 y 16 bits.

2.g. DIRECCIONAMIENTO DE PAGINA CERO MODIFICADO

La página 0 se define, en el caso del Z80, como el área de memoria externa que puede ser direccionado con 8 bits, es decir, los 256 bytes de memoria cuyas direcciones van desde 0000H hasta 00FFH.

Este modo de direccionamiento se utiliza solo para una instrucción, RST P (Restart Page Zero). El efecto de esta instrucción es causar un salto a alguna de 8 posiciones de página 0, luego de salvar el contenido actual del PC en el Stack.

Las 8 posiciones a que hacíamos referencia son las direcciones múltiplos de 8, empezando por la 0h.

Mnemónico	Opcode	función
RST P	1 1 p p p 1 1 1	PUSH PC; PC <-- P*8

2.h. DIRECCIONAMIENTO RELATIVO

Este modo de direccionamiento es utilizado, en el caso del Z80, solo por las instrucciones de salto (condicional o incondicional).

La instrucción suministra una constante que es un desplazamiento de 8 bits, en complemento a 2 que debe sumarse algebraicamente al PC para obtener la dirección donde continuar ejecutando el programa. El desplazamiento será pues un valor comprendido entre -128_{10} y $+127_{10}$.

Dado que la suma del PC con el desplazamiento se realiza cuando el PC está apuntando a la dirección que sigue a la instrucción de salto (es decir, se ha incrementado en 2), puedo saltar a posiciones de memoria que disten entre -126_{10} y $+129_{10}$ bytes de la instrucción de salto.

Ejemplo:
Consideremos la situación de la Fig. 6 en donde, en la posición 1200H de memoria se encuentra un opcode 28H, correspondiente a la instrucción "saltar si es cero", JR Z seguido de un byte que indica un desplazamiento

1200H	28 H
1	desplaz.
2	

Fig. 6

caso a:
desplaz = 01000000B = +40H

La dirección de la siguiente instrucción a ejecutar será:
1200H + 2 + 40H = 1242H.

caso b:

desplaz = 11110000B = -10H.

La dirección de la siguiente instrucción a ejecutar será:

1200H + 2 - 10H = 11F2H.

Como puede observarse, presenta la ventaja, frente al direccionamiento absoluto o directo, que la instrucción que lo utiliza ocupa un byte menos de memoria que la que utiliza el direccionamiento absoluto.

Sin embargo, la ventaja más importante de este modo de direccionamiento es que permite modificar la ubicación en memoria de un trozo de programa sin alterar el código de máquina. Nótese que para que esto sea posible, todos los saltos deben ser de este tipo.

2.i. DIRECCIONAMIENTO INDEXADO

Se utiliza junto a los dos registros índices del Z80, IX e IY. Muchas instrucciones permiten utilizar este modo de direccionamiento, que es una de las características más poderosas (y novedosas para su época) del Z80.

El formato de las instrucciones que utilizan este modo de direccionamiento es como sigue,

1er. byte: OPCODE
2do. byte: OPCODE
3er. byte: DESPLAZAMIENTO (8 bits, con signo)
4to. byte: (opcional)

El proceso para calcular la dirección efectiva (es decir, la dirección de memoria donde se encuentra el operando) es:

1. tomar el registro índice que corresponda (definido por el opcode)
2. sumar algebraicamente el desplazamiento

Ejemplos:

Mnemónico	Opcode	función
LD(IY + DESP),n	FD 36 DESP n	(IY + DESP) <-- n
RRC (IX + DESP)	DD CB DESP 0E	ROTAR A LA DER (IX + DESP)

Sea, en el primer caso, 1003H el contenido del registro índice IY, y sea DESP = 33H. La instrucción pondrá el valor "n" en la posición de memoria dada por IY + DESP = 1036H.

Esta operación de indexado es poderosa porque permite que los programas procesen tablas o listas de datos en memoria. Las instrucciones que utilizan este modo de direccionamiento son las de transferencia de datos de 8 bits, las aritméticas y lógicas de 8 bits, las de rotación y desplazamiento, las de bit, set, reset y test.

2.j. DIRECCIONAMIENTO A BIT

Se utiliza conjuntamente con los otros modos de direccionamiento para acceder a bits de registros o memoria, para setearlos, resetearlos y testear.

Ejemplo:

Mnemónico	Opcode	función
SET B,(IX + DESP)	DDH	Pone en 1 ("setea"), el
	CBH	bit B (0 <= B < 8) de
	DESP	la palabra de memoria dada
	1 1 B 1 1 0	por [IX + DESP].

El campo B ocupa tres bits, correspondiendo el código 000 al bit menos significativo.

Este modo de direccionamiento es muy práctico en situaciones en que se desea manejar líneas de control o monitorear líneas de estado individuales.

En los microprocesadores que no poseen esta capacidad se emplean operaciones lógicas bit a bit con constantes ("máscaras"). Por ejemplo, si se quiere setear el bit 2 del contenido del acumulador, sin alterar los demás bits, puede utilizarse la máscara 0000 0100$_2$, y ejecutar

OR A, 00000100B

Asimismo, para poner en cero el bit 5 del acumulador, se ejecutará

AND A, 11011111B

Si ahora quiere evaluarse el bit 0 del acumulador, puede hacerse

AND A, 00000001B

lo que afecta a las banderas, al igual que la instrucción TEST del Z80. Notar, sin embargo, que hemos destruido el contenido del acumulador.

3. Repertorio de instrucciones

3.a.FORMATO DE LAS INSTRUCCIONES

Las instrucciones pueden tener entre 1 y 4 bytes. Como por cada byte de instrucción se tendrá que hacer un acceso a memoria que, veremos, lleva 3 ó 4 períodos de reloj, las instrucciones que más lugar ocupan son también las más lentas.
La instrucción consta de :

- Un código de operación ("Opcode") que indica qué operación ha de realizarse (común a todas las instrucciones). Este campo puede tener una longitud de uno o dos bytes.

- En algunos caso, se requiere un byte de datos (direccionamiento inmediato, ver 2.b) o dos bytes (direccionamiento inmediato extendido, ver 2.c).

- En algunos casos, se requiere que en la propia instrucción se especifique una dirección, lo que implica 2 bytes adicionales (direccionamiento directo, relativo, etc).

Estudiaremos las instrucciones por grupos:

1- Transferencia, de 8 bits.
2- Transferencia, de 16 bits.
3- Intercambio, transferencia de bloques, búsqueda.
4- Lógicas y aritméticas de 8 bits.
5- Aritméticas de propósito general y control de CPU
6- Aritméticas de 16 bits.
7- Rotación y desplazamiento.
8- Bit Set, reset y Test.
9- Transferencia de control
10-Entrada y Salida

La descripción de las instrucciones que se realiza en este capítulo no pretende ser exhaustiva, ya que para ello existe muy buena bibliografía. Sin embargo, el manejo de todos las instrucciones que figuran aquí debería permitir al lector construir programas en forma aceptable.

3.b. INSTRUCCIONES DE TRANSFERENCIA DE DATOS DE 8 BITS.

Se trata de instrucciones que transfieren un dato de 8 bits desde un registro de la CPU, una posición de memoria, o un valor constante a un registro de la CPU y viceversa. EL contenido del origen nunca se ve alterado por una lectura.

Ejemplos:

a) LD r, r’   r <-- r’

Se trata de una operación de 1 byte, codificado así:

```
bit: 7 6  5 4 3  2 1 0
     0 1    r      r'
```

Tanto r como r' se especifican según la tabla 4. Obsérvese que el código 110 no está reservado para ningún registro. Fijándose en la cartilla, puede verse que se utiliza ese código, justamente, para indicar que el operando no es un registro.

r,r'	REGISTRO
000	B
001	C
010	D
011	E
100	H
101	L
111	A

4 - Campos de especificación de los registros.

b) LD r,CTE   r <-- CTE

aquí se requieren 2 bytes:

```
           bit: 7 6  5 4 3  2 1 0

1er. byte:      0 0    r    1 1 0
2do. byte:      V A L O R   C T E
```

Estas instrucciones pueden ser utilizada casi con la totalidad de los modos de direccionamiento. Sin embargo, debe cumplirse la regla general que indica que solo uno de los dos operandos puede residir en memoria externa. Dicho de otra manera, no existe la instrucción LD op1, op2, donde tanto op1 como op2 son posiciones de memoria.

3.c. INSTRUCCIONES DE TRANSFERENCIA DE DATOS DE 16 BITS.

Permite que cualquier par de registros BC, DE, HL o los registros SP, IX e IY sean cargados directamente con una constante, por ejemplo

LD IX, 48F3H

o que se realice una transferencia de 16 bits desde o hacia una dirección constante de memoria:

LD BC, (3DB1)

LD (1000), IY

B	AB	CD	C
D			E
H			L
IX	48F3		
IY	1234		

Dirección	Contenido
1000 H	34
1001 H	12
.	.
.	.
.	.
3DB1 H	CD
3DB2 H	AB

En el recuadro, se observa el estado de los registros y la memoria luego de efectuadas las tres transferencias ejemplificadas.

Dentro de este grupo, también encontramos las instrucciones de STACK, que, como se dijo, siempre son de 16 bits: puede hacerse POP qq o PUSH qq, donde se guardan los pares, de acuerdo a la tabla.

Observar que siempre el acumulador y el registro de estado se guardan conjuntamente.

En cuanto a transferencias directas entre registros de 16 bits, solo esta permitido cargar SP con HL, IX o IY.

qq	PAR
00	BC
01	DE
10	HL
11	**AF**

3.d. INTERCAMBIO, TRANSFERENCIA DE BLOQUES Y BUSQUEDA.

Las instrucciones de intercambio permiten:

- Intercambiar datos entre registros de 16 bits de un mismo banco de registros.

EX DE, HL

- Intercambiar acumulador y palabra de estado inactivo por activo:

EX AF, AF'

- Intercambiar banco de registros de uso general activo e inactivo:

EXX

- Intercambiar el último dato del stack con HL, IX e IY:

EX (SP), IX

Estudiemos este último ejemplo con más detalle:

ANTES:

SP	3579
IX	ACE0

DESPUES:

SP	3579
IX	6824

Contenido de memoria:

ANTES:

3579	24
357A	68

DESPUES:

3579	E0
357A	AC

Dentro de las instrucciones de transferencia de bloques[3], LDI, LDIR, LDD Y LDDR, se utilizan los siguientes registros:

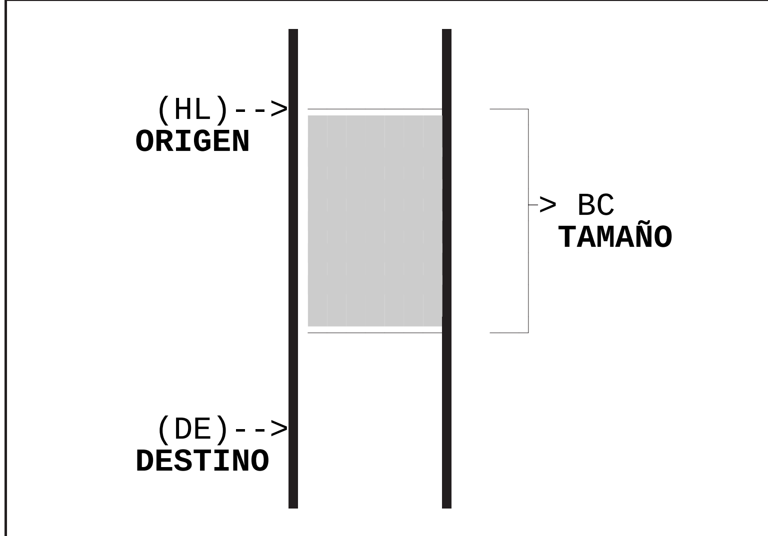
BC : número de bytes a transferir (hasta 64 K)
HL : dirección de comienzo en origen del bloque a transferir
DE : dirección de comienzo en destino.

[3] - En otros microprocesadores no existen estas instrucciones, y para lograr el mismo efecto se utiliza un conjunto de instrucciones más sencillas.

Al ejecutar LDI (LoaD and Increment) se producen las siguientes acciones:

1 - (DE) <-- (HL)
2 - DE <-- DE + 1
    HL <-- HL + 1
3 - BC <-- BC - 1
4 - si BC <> 0, la bandera P/V = 1

LDD = LoaD and Decrement, es igual, salvo que deben cambiarse los signos + en el paso 2 por -.



6 - Uso de registros en transferencias de bloques

Las instrucciones LDIR y LDDR son respectivamente iguales a LDI y LDD, pero el punto 4 cambia por el siguiente:

4 - si BC <> 0, vaya al paso 1.

por lo que automatizan totalmente la transferencia de un bloque desde una posición de memoria a otra.

Las instrucciones de búsqueda, son similares a las de transferencia y permiten detectar en qué posición de memoria se encuentra un byte "clave". Utilizan los siguientes registros:

A: Contiene la "clave"
BC: Tamaño del bloque, como en el caso anterior.
HL: Dirección de comienzo.

Cuando se ejecuta CPI, se realizan las siguientes acciones:

1- Se accede al byte apuntado por HL y se lo compara con A: si ambos bytes son iguales, se setea la bandera Z.
2- HL <-- HL + 1
3- BC <-- BC - 1

Nuevamente, si se desea decrementar el puntero, se debe utilizar CPD.

Las instrucciones CPIR y CPDR realizan reiteradamente lo mismo que CPI y CPD, hasta que BC = 0 o Z = 1.

3.e. INSTRUCCIONES LOGICAS Y ARITMETICAS DE 8 BITS.

Se utilizan para sumar, restar, comparar, hacer AND, OR, XOR entre 2 operandos de 8 bits, uno de los cuales está en el acumulador.

Existen 2 variedades de suma: ADD y ADC. La primera es la suma común de dos datos de 8 bits. La segunda, además de la suma entre los 2 operandos, suma el contenido de la bandera de Carry: esto es útil para trabajar con números de más de 8 bits.

Consideremos el siguiente ejemplo: Se desea sumar los números hexadecimales de 16 bits siguientes:

```
               71 8A   +  43 91

Ubicados en      H  A       B  C
Registro
```

1º - Sumo los bytes bajos de cada uno:

```
ADD C          A <-- A + C
```

El resultado de esta suma será 8A + 91 = 1B, y se seteará la bandera de carry, es decir CF = 1.

2º - A los efectos de dejar libre el acumulador, muevo el resultado al registro C (notar que esta instrucción no afecta las banderas). Luego muevo uno de los bytes altos al acumulador.

```
LD C, A        C <-- A
LD A, H        A <-- H
```

3º - Efectúo la suma de los bytes altos, sumando además el carry:

```
ADC B.         A <-- A + B + CF
```

El resultado será, 71 + 43 + 1 = B5, con lo que se obtiene la suma de dos números de 16 bits: B51B H.

Observar que, como ADC afecta también la bandera CF, se puede repetir este procedimiento a los efectos de sumar números de mayor número de bits.

Lo mismo es cierto para "el préstamo" de la resta, por lo cual existen dos restas, la común SUB y SBC, que efectúa la resta común y al resultado le resta el contenido anterior del carry.

Una instrucción muy interesante es la de <u>comparación.</u> Si efectuamos:

```
CP operando_8_bits
```

el microprocesador efectúa la resta, A - operando_8_bits, y afecta las banderas de cero, signo, acarreo y H, *pero descarta el resultado de la resta, es decir, no lo guarda.* Como veremos en los ejemplos, más adelante, esto es muy práctico para testear condiciones.

Este grupo de instrucciones se completa con las operaciones lógicas habituales (AND, OR, XOR) y las de incremento y decremento (INC y DEC).

3.f. INSTRUCCIONES ARITMETICAS DE PROPOSITO GENERAL Y CONTROL DE CPU

Se trata de instrucciones que todas utilizan direccionamiento implícito, involucrando uno o ningún operando.

* Operaciones de complemento del Acumulador

- NEG : Complemento a 2 del acumulador.
- CPL : Complemento lógico (complemento a 1)

Ejemplo:

```
     A = 10010110  ( corresponde a -6AH, en comp. a 2 )
CPL
     A = 01101001
NEG
     A = 10010111  ( corresponde a -69H, en comp. a 2 )
NEG
     A = 01101001  ( corresponde a +69H, en comp. a 2 )
```

* Existen instrucciones referentes a la bandera de carry:

- SCF : Setea C = 1;
- CCF : invierte el valor de C.

* Otras operaciones:

NOP : es una instrucción que no hace nada: sirve para, por ejemplo, "rellenar" un bucle de tiempo.

HALT: Es una instrucción para detener al procesador. Típicamente, se utiliza esta instrucción con dos finalidades. Una posibilidad, es que el programa haya realizado todas sus tareas, y la otra, es que el programa deba esperar por una interrupción.

DI, EI : deshabilita y habilita interrupciones enmascarables.

IM0, IM1, IM2 : Se refieren al modo de interrupción, que se verá después.

3.g. INSTRUCCIONES ARITMETICAS DE 16 BITS.

Cuando se hable de registros de 16 bits, se está excluyendo al PC.

Permiten:

- Decrementar o incrementar todos los registros de 16 bits y los pares BC, DE Y HL.

- Sumar, sumar con carry, restar con préstamo entre varias combinaciones de registros. No existe la resta común entre registros de 16 bits.

3.h. ROTACION Y DESPLAZAMIENTO.

Este grupo de instrucciones tiene por finalidad la manipulación de los bits de un registro o una posición de memoria.

Por su naturaleza, estas instrucciones se explicitan mejor en forma gráfica.

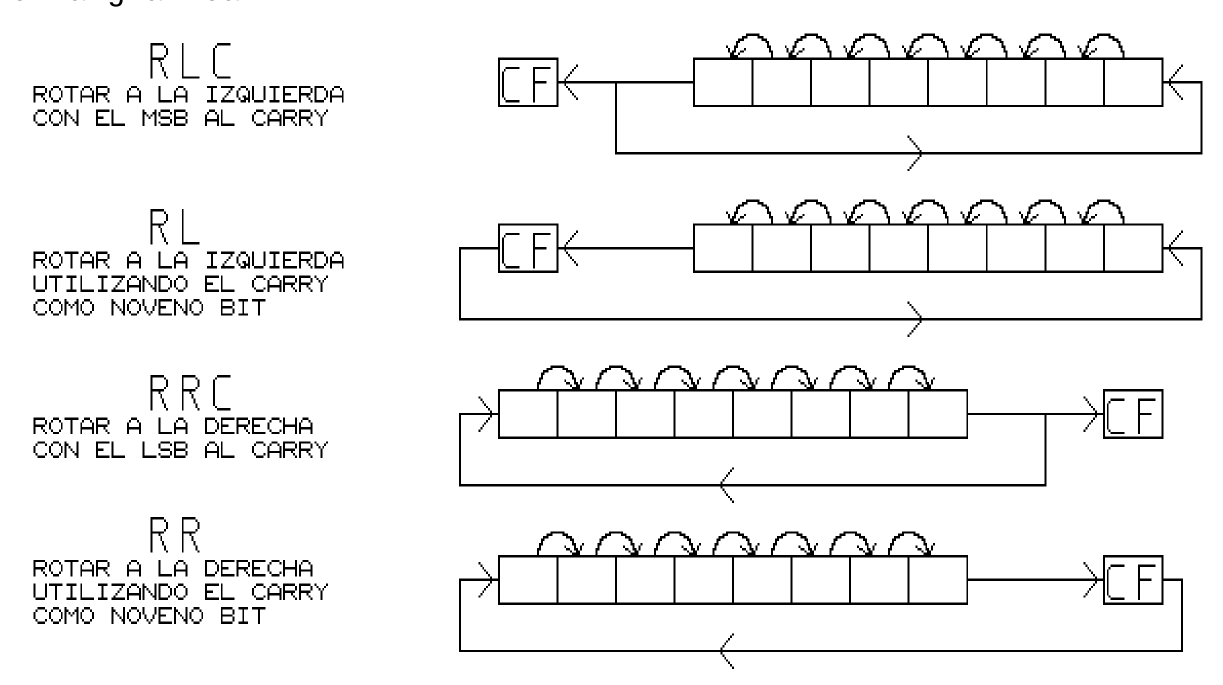


Fig. 7 Instrucciones de Rotación.

En cuanto al significado de los mnemónicos, la R inicial es de "rotate", la segunda letra corresponde al sentido: "Left" o "Right". Y la tercera, si existe es una "C", y se refiere a la participación de la bandera de carry en la operación.

Observando las tablas de instrucciones del apéndice, se verificará que existen operaciones "repetidas". Por ejemplo, la rotación circular a la izquierda del acumulador puede realizarse con la instrucción de un byte RLCA, o con la de dos bytes RLC A. Esto se debe a que el conjunto de instrucciones del Z80 es un superconjunto de las instrucciones de otro procesador más antiguo, el 8080, que solo permitía la manipulación de bits del acumulador.

Existen un par de instrucciones que realizan "rotaciones", pero en lugar de trabajar de a un bit, trabajan con conjuntos de a 4 bits, y están orientadas a la implementación de operaciones con número en BCD. Ellas se muestran en la Fig. 8

En cuanto a las instrucciones de desplazamiento, sus mnemónicos comienzan con la letra "S" (shift). En la Fig. 9 están representadas las operaciones de desplazamiento.

En el desplazamiento aritmético a la derecha (SRA) se repite el bit de signo, lo cual equivale a dividir por 2 un número en

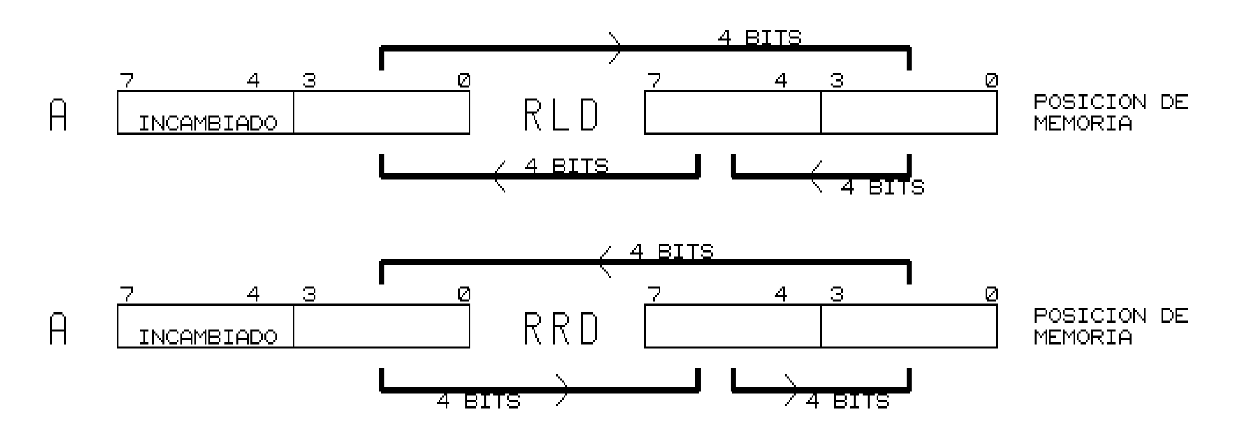


Fig. 8

complemento a 2.

En el desplazamiento aritmético a la izquierda (SLA) se rellena con un cero, lo que implica multiplicar por 2.

El desplazamiento lógico a la derecha puede verse también como una división por dos en el caso de enteros sin signo.

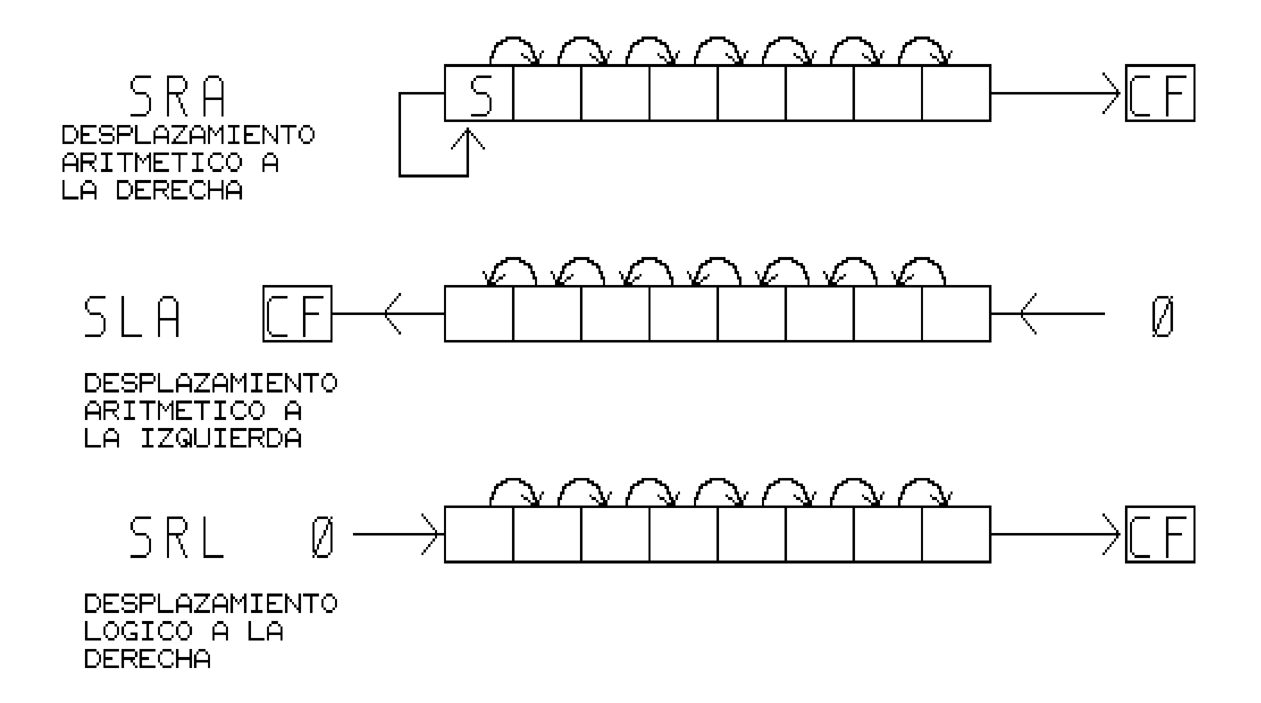


Fig. 9 - Instrucciones de desplazamiento

3.i. BIT SET, RESET Y TEST.

Existen tres operaciones básicas a ser realizadas sobre bits. Estos bits pueden corresponder a uno de los registros de 8 bits de

la CPU o a un operando de memoria.

La operación

BIT B, R

donde B es un No. del 0 al 7 y R es un registro, setea la bandera Z si el bit B-ésimo de R es 0 y la resetea en caso contrario.

También pueden setearse o resetearse bits mediante SET, y RES:

RES 4, (HL)

Supongamos que HL = 2345, y que (2345) = F0, luego de la operación será (2345) = E0.

3.j. INSTRUCCIONES DE TRANSFERENCIA DE CONTROL

Pueden dividirse en saltos (Jumps), Llamados a subrutinas (Calls) y retornos (Returns).

Los saltos causan la transferencia de control a otra posición de memoria y no guardan el contenido anterior del PC.

Los llamados a subrutinas realizan la misma acción que los jump, pero salvan el PC en el stack de modo de que al retornar se ejecute la instrucción que sigue al call.

El Return realiza el "pop" del PC desde el Stack, provocando que se ejecute la instrucción que estaba a continuación del call en el programa principal.

Las instrucciones CALL Y RET se utilizan para procesar subrutinas. Las subrutinas son un conjunto de instrucciones que se utilizan varias veces en un programa, y que está escrito en un solo lugar. Si el programador desea intercalar una subrutina en su programa, la invoca, utilizando "CALL". Como la subrutina termina con RET, luego de ejecutarse, continuará ejecutándose el programa que la invocó. Volveremos sobre este tema más adelante.

<u>Salto incondicional</u>

a) Absoluto: JP nn : la próxima instr. a ejecutar es la que está en la dir. nn.

b) Relativo: JR despl.: la próxima instr. a ejecutar es la que está en la dir PC + despl.

<u>Salto condicional</u>

a) absoluto: JP cc, nn: si cc es cierto, PC <-- nn; el valor de cc es según tabla que se adjunta.

cc puede ser:

NZ (no cero)	NC (sin carry)	PO (paridad impar)	NS(positivo)
Z (cero)	C (carry)	PE (paridad par)	S (negativo)

b) relativo: JR cc, nn, pero ahora cc solo puede ser:
NZ, Z, NC, C.

Decremento y salto

La instrucción DJNZ despl, decrementa B, y si el resultado no es cero, salta. Si B = 0, no salta.

Subrutinas

Existe llamado incondicional, CALL nn, y condicional CALL cc,nn; lo mismo ocurre con el retorno: existe RET y RET cc; el significado de cc es el mismo que para el JP.

A los efectos de resumir todas las instrucciones que preguntan por el estado de banderas, se presenta la siguiente tabla:

BANDERA	SALTO ABSOLUTO	SALTO RELATIVO	LLAMADO A SUBRUTINA	RETOR-NO
Z=0 (no cero)	JP NZ,POS	JR NZ,REL	CALL NZ,RUT	RET NZ
Z=1 (= cero)	JP Z,POS	JR Z, REL	CALL Z,RUT	RET Z
C=0 (no carry)	JP NC,POS	JR NC,REL	CALL NC,RUT	RET NC
C=1 (hay carry)	JP C,POS	JR C,REL	CALL C,RUT	RET C
P/V=0 (paridad im-par, no overflow)	JP PO,POS	**NO EXISTE**	CALL PO,RUT	RET PO
P/V=1	JP PE,POS	**NO EXISTE**	CALL PE,RUT	RET PE
S=0 (>= 0)	JP P,POS	**NO EXISTE**	CALL P,RUT	RET P
S=1 (< 0)	JP M,POS	**NO EXISTE**	CALL M,RUT	RET M

3.k. INSTRUCCIONES DE ENTRADA Y SALIDA

El Z80 tiene un espacio de direccionamiento de I/O de 256 bytes. Puede transferirse datos entre cualquiera de las direcciones de I/O y los registros A, B, C, D, E, H y L. Para ello existen las instrucciones IN y OUT.

También existe la posibilidad de realizar transferencias de bloques de hasta 256 bytes entre memoria e I/O. Estas instrucciones son análogas a las de transferencia de bloques en memoria, con la única diferencia de que no se incrementa la dirección de E/S. Es decir, permiten transferir un bloque de datos desde o hacia un determinado puerto.

4. Software

4.a. NECESIDAD DEL ENSAMBLADOR. LENGUAJES DE PROGRAMACION.

El único lenguaje que entiende un procesador es la secuencia de "0" y "1" que lee de su memoria de programa e interpreta como instrucciones de su repertorio. A esa secuencia de "0" y "1" se le llama código de máquina de un programa. El paso final en el desarrollo de un programa es obtener el programa correcto, en código de máquina, cargado en memoria para que pueda ser ejecutado por el procesador.

La tabla Tabla III muestra el código de máquina correspondiente a un segmento de programa sencillo para el procesador Z80. El programa toma el contenido de dos direcciones de memoria, los suma y carga el resultado en otra dirección de memoria.

Tabla III- Código de máquina

0011 1010
0000 1100
0000 0000
0100 0111
0011 1010
0000 1101
0000 0000
1000 0000
0011 0010
0000 1110
0000 0000
0111 0110

Es evidente la ilegibilidad del código de máquina por parte de un lector humano. Esto hace que sea imposible desarrollar un programa, incluso de complejidad pequeña, directamente en lenguaje de máquina. Mucho más difícil todavía resulta entender un programa escrito antes para corregirlo o modificarlo. Necesitamos lenguajes que permitan un mayor nivel de abstracción, de manera de tener una representación más legible del programa.

Este mayor poder de abstracción se logra en los llamados LENGUAJES DE ALTO NIVEL. Tomando como ejemplo el PASCAL conocido por el lector, el segmento de programa del ejemplo se reduce a una sentencia:

```
Suma := Oper1 + Oper2;
```

Los lenguajes de alto nivel están definidos de manera abstracta, independientemente de qué procesador tenga el sistema en el cual se va a ejecutar el programa. Para poder ejecutar algo es necesario TRADUCIR el programa escrito en lenguaje de alto nivel a código de máquina particular del procesador en que se va a ejecutar el programa para hacerlo comprensible por el mismo. A ese proceso de traducción se le llama COMPILACION.

La compilación se realiza usualmente en forma automática utilizando un programa COMPILADOR. Este toma como entrada el programa original escrito en alto nivel, llamado programa fuente, y genera el código de máquina correspondiente al que se le llama programa objeto.

En un nivel de abstracción por encima del código de máquina pero por debajo de los lenguajes de alto nivel tenemos al LENGUAJE ENSAMBLADOR o ASSEMBLER. Aquí a cada instrucción del procesador le

corresponde una abreviación mnemotécnica o MNEMONICO que la hace más comprensible por el programador. No es necesario memorizar el patrón de "0" y "1" con que se codifica una instrucción y podemos entonces concentrarnos en la función que queremos realizar.

A diferencia de los lenguajes de alto nivel, el assembler nos permite controlar todos los recursos del hardware que estamos usando. Asimismo los programas escritos en assembler pueden ser muy eficientes tanto en tiempo de ejecución como en tamaño. Puede suceder también que para el sistema que estamos programando no exista un compilador de alto nivel. Por todas estas razones puede ser preferible escribir un programa en lenguaje ensamblador a pesar de su menor nivel de abstracción comparado con los lenguajes de alto nivel.

El lenguaje ensamblador es específico de un determinado procesador. Las abreviaciones mnemotécnicas (mnemonics en inglés) las elige el fabricante del procesador y corresponden a cada una de sus instrucciones.

Se permite además definir símbolos para identificar direcciones de memoria o valores utilizados a menudo a lo largo del programa.

El segmento de programa mostrado en código de máquina en la tabla Tabla III se reduce al segmento de programa en assembler de la columna de la derecha. La otra columna contiene los bytes de código de máquina correspondientes a cada instrucción, mostrados esta vez en código hexadecimal.

```
...
   3A 0C 00                LD A,(OPER1)
   47                      LD B,A
   3A 0D 00                LD A,(OPER2)
   80                      ADD A,B
   32 0E 00                LD (SUMA),A
   76                      HALT
....
```

Es necesario también aquí realizar una traducción pero mucho más sencilla ya que a cada mnemónico le corresponde una instrucción en código de máquina. En este caso particular, a la traducción se le llama ENSAMBLADO. También se hace usualmente en forma automática y al programa que la realiza se le llama ENSAMBLADOR o ASSEMBLER (igual que al lenguaje).

4.b. PROCESO DE DESARROLLO DE PROGRAMAS.

Si bien podría pensarse que el uso de una computadora no es indispensable para desarrollar un programa correcto en lenguaje ensamblador, las ventajas que esto proporciona hacen impensable -hoy en día- el desarrollo de programas sin su auxilio.

Se le llama AMBIENTE DE DESARROLLO a un conjunto de herramientas de software y hardware que facilitan el proceso de

desarrollo de programas desde el planteo inicial del problema a resolver hasta obtener el programa correcto cargado en la memoria de programa del microprocesador.

En esta sección pasaremos revista a esa gama de herramientas de ayuda analizando el campo de aplicación de cada una de ellas.

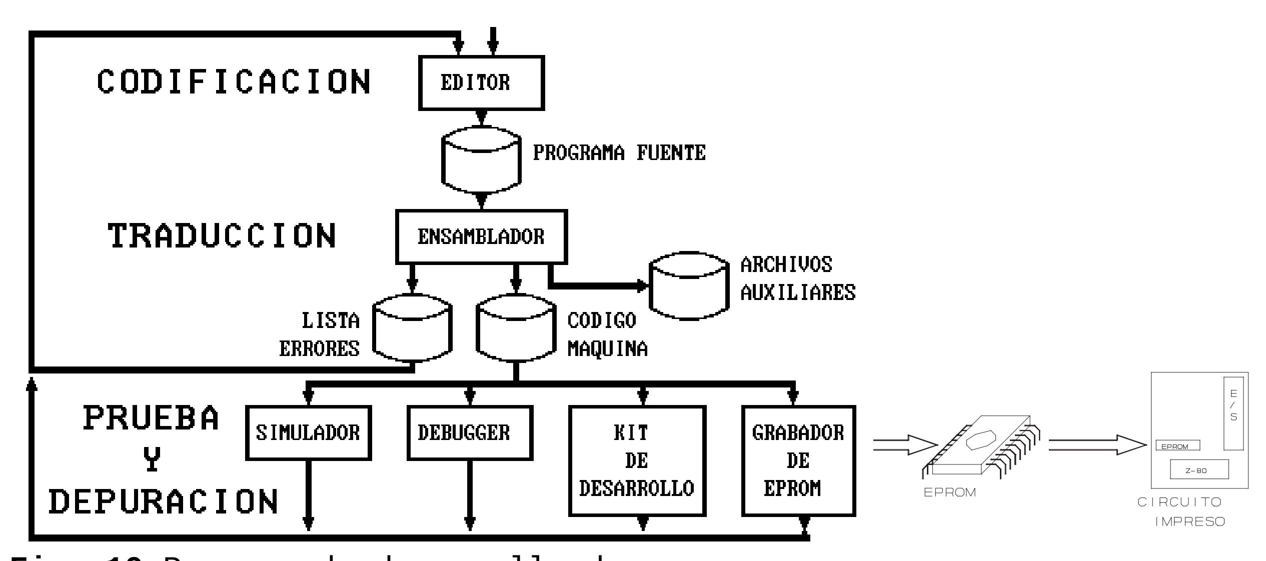


Fig. 10 Proceso de desarrollo de un programa.

Las herramientas a utilizar dependen de cuál sea el sistema en el cual se va a usar el programa. En este sentido existe una amplia variedad de posibilidades. En un extremo, puede suceder que el programa vaya a ejecutarse en un computador ya diseñado como puede ser un computador personal. En el otro extremo, podemos estar escribiendo el programa que va a controlar el funcionamiento de una lavadora y que va a ejecutarse en un computador que consiste en una plaqueta de circuito impreso con un Z80, una EPROM y puertos de entrada y salida y que ni siquiera ha sido aún construido. El proceso de desarrollo se ilustra en la Fig. 10.

En todos los casos habrá una etapa inicial de DISEÑO del programa en que planteamos el problema en algún lenguaje informal de alto nivel. Para ello se utilizan o bien los DIAGRAMAS DE FLUJO o bien describimos el programa utilizando estructuras de control similares a las de los lenguajes de alto nivel (while, if) pero tomándonos ciertas libertades para describir los recursos de nuestro sistema en particular. A esta última descripción se le llama a veces PSEUDOCODIGO del programa. El lector utilizará el método que le resulte más cómodo.

Una vez diseñado el programa debemos escribir el código del mismo. A esto se le llama CODIFICACION y el resultado, en nuestro caso el programa escrito en assembler, es el programa fuente.

Para auxiliarnos al escribir el programa se utiliza un EDITOR DE TEXTO. Este nos permite introducir un texto a través de un terminal con teclado y pantalla y almacenarlo como un archivo en

código ASCII en alguna memoria auxiliar del computador (disco flexible o disco rígido p. ej.).

Luego debemos realizar la traducción o ensamblado para obtener el código de máquina del programa. Esto se hace usualmente en forma automática con un ENSAMBLADOR. Este toma como entrada el archivo con el programa fuente que debe estar escrito respetando ciertas reglas de sintaxis del ensamblador. El ensamblador genera en primer lugar un listado con las violaciones a esas reglas. Si existen errores de sintaxis debemos corregir el programa fuente usando nuevamente el editor. Recién cuando el ensamblador no detecta errores gramaticales está en condiciones de generar el programa objeto. Opcionalmente permite generar además listados auxiliares mostrando el código de máquina correspondiente a cada línea del programa fuente y lo que se llama la tabla de símbolos, esto es, los valores asociados a cada símbolo definido en el programa.

Nótese que el computador utilizado hasta ahora, en el cual corren el programa editor y ensamblador no necesita ser el computador donde va a ejecutarse el código de máquina resultante (piénsese en el caso de la lavadora). Es más, ni siquiera es necesario que tenga el mismo procesador. Así por ejemplo, podemos editar y ensamblar los programas para el controlador de la lavadora trabajando en un computador personal AT compatible, cuya CPU es un microprocesador 80286 de INTEL, si disponemos de un programa editor y un programa ensamblador para el microprocesador Z80.

Una vez llegados a este punto estamos seguros que el programa no tiene errores de sintaxis. Esto es, nuestro programa será interpretado "de alguna manera" por el ensamblador. Pero no sabemos si no hay errores en la lógica del programa. No sabemos si el algoritmo que escribimos resuelve nuestro problema. Para determinar eso debemos hacer la PRUEBA y DEPURACION del programa.

Hay muchas alternativas para la etapa de prueba dependiendo del ambiente usado para el desarrollo y de en que computador vaya a correr el programa, pero básicamente consiste en lograr una ejecución del mismo con valores de entrada conocidos para determinar si se obtienen los resultados esperados.

Si no disponemos del sistema destino en el cual va a funcionar el programa la alternativa para hacer la prueba y la depuración es utilizar un SIMULADOR. Este es un programa que corre en otro computador y emula el funcionamiento del microprocesador. Utiliza lugares de memoria del computador para simular los registros del microprocesador y la memoria del sistema destino. El simulador lee cada instrucción del código de máquina y simula la ejecución de la misma modificando de manera adecuada los lugares de memoria del computador que simulan los "registros" y la memoria del sistema destino.

Por cada instrucción del código de máquina a simular, el simulador debe ejecutar varias instrucciones de su propio procesador. La velocidad a la que se ejecuta la simulación difiere por esta razón de la velocidad de ejecución real. Por lo tanto la simulación no permite ensayar el comportamiento del programa frente

a fenómenos vinculados con la variable tiempo.

Un DEPURADOR o DEBUGGER es un programa que corre en un computador con un microprocesador igual al del sistema destino y permite realizar una ejecución controlada de nuestro programa. Podemos ejecutar el programa de a una instrucción por vez o hasta la ocurrencia de algún evento especificado, deteniendo en ese caso la ejecución y permitiendo examinar y modificar el contenido de los registros del microprocesador o lugares de memoria del sistema en el momento de detenerse la ejecución.

Los eventos que podemos especificar para que provoquen la detención del programa pueden ser varios pero el más común en los depuradores es un "punto de ruptura", o en inglés BREAKPOINT. Podemos marcar como breakpoint una línea de nuestro programa y la ejecución se detendrá justo antes de ejecutarse esa instrucción. Esto nos permite seguir de cerca la ejecución del programa y así detectar errores molestos y difíciles de encontrar a los que se les llama usualmente BUGS[4].

Los simuladores en general incluyen facilidades similares a las que brindan los depuradores.

A diferencia de una simulación, en este caso nuestro programa es ejecutado realmente en un microprocesador igual al que vamos a usar en el sistema destino. Por esta razón la depuración realizada con una herramienta de este tipo es mucho más potente que una simulación.

Si el sistema destino es un computador ya construido y disponemos de un debugger para el mismo podemos completar la depuración del programa ya que lo estamos testeando en condiciones similares a las del funcionamiento normal.

En otros casos se utiliza lo que se llama un sistema de desarrollo o KIT DE DESARROLLO. Esto consiste en un computador montado sobre una plaqueta controlado por un programa MONITOR. Este programa permite cargar el código de máquina a la memoria RAM del sistema y ejecutarlo de manera controlada en forma similar a un debugger. La CPU es un microprocesador igual al del sistema destino, y la plaqueta tiene lugar disponible a efectos de probar diferentes configuraciones de hardware en las etapas de desarrollo del equipo.

En caso de no disponer de un sistema de desarrollo con un monitor deberemos construir un prototipo hardware de nuestro sistema final. Parte del programa (y a veces todo) residirá en ROM. Para probar nuestro programa (y también el hardware) debemos cargar el código de máquina en ROM mediante un equipo GRABADOR DE EPROM para posteriormente insertar el chip de memoria en la plaqueta del prototipo. Téngase presente que en este caso el programa debe

[4]Bug podría traducirse como "pulga". Uno de los debuggers más difundidos para microprocesadores de 8 bits se llama "Dynamic Debugging Tool", más conocido por la sigla DDT.

incluir la rutina de inicialización del microprocesador que se ejecuta luego de un RESET hardware y la ejecución del programa comenzará precisamente al conectar la alimentación al prototipo.

Existen otras herramientas más potentes para depurar el software y el hardware de un sistema basado en microprocesador como son los EMULADORES EN CIRCUITO y los ANALIZADORES pero en esta introducción al tema no las veremos en detalle.

En todos estos casos, cada vez que detectemos un error en el programa debemos corregirlo en el programa fuente en assembler con la ayuda del editor y repetir el proceso.

En un desarrollo típico se pueden usar varias de las herramientas que vimos. Por ejemplo, si escribimos el programa para una plaqueta basada en microprocesador podemos hacer una primera depuración utilizando un simulador donde eliminamos buena parte de los errores. De esta manera partimos de un programa razonablemente libre de errores y de esa forma tendremos que realizar menos veces el proceso más lento de grabar una EPROM.

Una vez terminado el proceso de depuración tenemos el código de máquina correcto, listo para ser cargado en la memoria de programa del computador destino. Un programa CARGADOR transfiere código de máquina desde algún medio de almacenamiento externo (un archivo en disco flexible p. ej.) a la memoria de programa del sistema y le transfiere el control. Si nuestro sistema destino es un computador personal, esta función es realizada por el sistema operativo de la máquina. Si estamos desarrollando un programa para una tarjeta con microprocesador el programa es cargado en ROM mediante el grabador de EPROM como en la etapa de depuración.

4.c. LENGUAJE ENSAMBLADOR.

Como ya vimos, en lenguaje ensamblador tenemos un mnemónico para cada una de las instrucciones del repertorio de instrucciones de un microprocesador dado. Existen por lo tanto al menos tantos lenguajes ensamblador como microprocesadores.

En esta sección analizaremos en detalle las tareas que se deben realizar para hacer la traducción de lenguaje ensamblador a código de máquina y veremos las reglas de sintaxis y directivas más usuales, comunes a la mayoría de los ensambladores.

Un programa en assembler está formado por una secuencia de sentencias. Cada sentencia ocupa una línea del programa y en general puede ser de dos tipos:

* Instrucción. Una sentencia de este tipo se traducirá como una instrucción del microprocesador. Estará formada por el mnemónico de la instrucción y, en el caso que corresponda, el o los operandos.

* Directiva al ensamblador o seudoinstrucción. Completa la información que necesita el ensamblador respecto a las

direcciones de memoria utilizadas por el programa y definición de símbolos que veremos más adelante en detalle.

Además de manipular los mnemónicos de cada instrucción del microprocesador, el lenguaje ensamblador permite utilizar identificadores para designar constantes o direcciones a lo largo del programa.

Antes de describir en forma detallada todos estos elementos y las directivas al ensamblador hagamos manualmente la traducción de un programa sencillo en assembler a código de máquina. Esto nos permitirá comprender más fácilmente los procesos involucrados en la traducción automática realizada por un ensamblador.

4.c.I.ENSAMBLADO MANUAL. UN EJEMPLO.

En el recuadro 7 tenemos el código fuente en lenguaje ensamblador de un programa sencillo. Se utilizan símbolos para referir a constantes como en el caso de CANTIDAD, o a direcciones de memoria o E/S como en los demás símbolos usados. El objetivo de la traducción será sustituir los mnemónicos por los códigos de operación, y los demás símbolos por la constante binaria que corresponda.

```
DIRECC CODIGO     ETIQUETA  INSTRUCCION
       MAQUINA

                  RETARDO:  LD   B, CANTIDAD
                            LD   A, 1
                            OUT  (SALIDA), A
                  LAZO:     DEC  B
                            JP   NZ, LAZO
                            LD   A, 0
                            OUT  (SALIDA), A
```

7-Programa fuente en lenguaje ensamblador.

A esos efectos se reservan dos columnas a la izquierda del programa fuente. En la primera pondremos la dirección de memoria donde debe cargarse el primer byte de cada instrucción. En la segunda pondremos los bytes de código de máquina generados para cada instrucción, representados en forma hexadecimal para mayor claridad.

El ensamblado se lleva a cabo en dos etapas, cada una de las cuales requiere una recorrida completa o pasada por todo el programa fuente. En la primer pasada se determina la posición en memoria donde se cargará el primer byte de código de cada instrucción y por otra parte se crea una tabla donde se anota el valor asociado con cada símbolo que aparece en el programa. Para eso se asigna la dirección inicial u origen al primer byte de la primera

instrucción y lo anotamos en su lugar en la primer columna. A partir de allí llevamos un conteo incrementando en la cantidad de bytes de cada instrucción y anotando el resultado en el siguiente lugar en la columna de direcciones. A dicho conteo se le llama CONTADOR DE POSICION.

```
 DIRECC CODIGO     ETIQUETA  INSTRUCCION
        MAQUINA

                                                 TABLA DE
  0000             RETARDO: LD   B, CANTIDAD       SIMBOLOS
  0002                      LD   A, 1
  0004                      OUT  (SALIDA), A  LAZO    0006H
  0006             LAZO:    DEC  B            RETARDO 0000H
  0007                      JP   NZ, LAZO     SALIDA  0010H
  000A                      LD   A, 0         CANTIDAD  32H
  000C                      OUT  (SALIDA), A
```

8-Pasada 1

La etiqueta, de las instrucciones que la tienen como p. ej. LAZO, se anota en la tabla de símbolos haciéndole corresponder la dirección del primer byte de la instrucción que ahora tenemos anotada en la columna de direcciones. Estas etiquetas son nombres simbólicos utilizados como direcciones en varias instrucciones del programa. Además de las direcciones que se definen como etiquetas, se escriben simbólicamente otros operandos, pero sus valores deben definirse en forma explícita. Estos nombres simbólicos y sus valores se agregan también en la tabla de símbolos. El resultado de la primer pasada se muestra en el recuadro 8.

En la segunda pasada se obtiene el código de máquina. Se examina cada instrucción y se coloca el código de operación y los operandos que correspondan en su lugar en la segunda columna. Para hacer esto manualmente será necesario consultar la cartilla de instrucciones del microprocesador para obtener el código de operación. Cada vez que aparezca un nombre simbólico entre los operandos deberemos consultar en la tabla de símbolos generada en la primera pasada el valor binario que tiene asociado.

Al final de la segunda pasada la traducción está completa y el código de máquina está en la segunda columna como se muestra en el recuadro 9.

El procedimiento descrito permite realizar la traducción de un programa contenido en un único módulo. Esto es suficiente para muchos programas sencillos. Un método más complejo permite trabajar con varios módulos de programa. En este caso el ensamblador hace una primera traducción de cada módulo dejando pendiente para un proceso posterior, llamado LINKER, todo lo que no puede resolver. El LINKER toma todos los módulos ensamblados y genera el código de máquina final.

```
 DIRECC CODIGO     ETIQUETA  INSTRUCCION
        MAQUINA

  0000  06 32      RETARDO:  LD   B, CANTIDAD
  0002  3E 01                LD   A, 1
  0004  D3 10                OUT  (SALIDA), A
  0006  05         LAZO:     DEC  B
  0007  C2 06 00             JP   NZ, LAZO
  000A  3E 00                LD   A, 0
  000C  D3 10                OUT  (SALIDA), A
```

9-Pasada 2

4.c.II. REGLAS DE SINTAXIS Y DIRECTIVAS AL ENSAMBLADOR.

Se describen a continuación los elementos más usuales del lenguaje ensamblador.

* Identificadores.

Son cadenas de caracteres que una vez definidas se pueden utilizar para designar diferentes constantes. En general los ensambladores exigen que tengan una longitud limitada (menor que 8 caracteres p. ej.) y que no comiencen con un dígito. Serían identificadores válidos en ese caso p. ej. OPER1, fin_prog, LAZO_01 y no lo sería 01_LAZO.

* Constantes.

Representan valores numéricos o caracteres.

Las constantes numéricas en general se acepta que se escriban por lo menos como números decimales, binarios o hexadecimales. Usualmente el ensamblador interpreta por defecto que se trata de un número decimal debiendo agregarse algún carácter para indicar que es un número binario o hexadecimal. Sea como sea que las escribimos, estas constantes serán convertidas por el ensamblador en un número binario de la cantidad de bits del registro que corresponda del microprocesador P. ej.:

```
ADD A,5
LD (5),A
```

En el primer caso el 5 será interpretado como un número binario de 8 bits para ser sumado al acumulador. En el segundo caso se convertirá en un número de 16 bits que da la dirección en memoria a la que se transfiere el contenido del acumulador. Para el hexadecimal se exige que comience con un dígito para distinguirlo de un identificador. Se ilustra con un ejemplo las notaciones más usuales.

Decimal	253		
Binario	11111101B	ó	$11111101
Hexadecimal	0FDh	ó	#FD

Las constantes de caracteres se escriben entre comillas simples y se interpretan en gral como el código ASCII correspondiente al carácter. P. ej.: ’a’ se interpreta como el número binario de 8 bits 0110 0001.

* Contador de posiciones $.

Se reserva un identificador especial, generalmente el símbolo $, para referirnos al lugar de la memoria en que debe cargarse el primer byte de código generado para cada línea de programa. Es una variable interna del ensamblador. Se inicializa mediante la directiva ORG y se va incrementando al ensamblar cada línea de programa en la cantidad de bytes que ocupa el código generado al traducir esa línea.

* Expresiones.

Los argumentos de una instrucción o seudoinstrucción son una expresión que podrá contener constantes e identificadores relacionados por operadores unarios (+, -, NOT) u operadores binarios (+, -, *, /, AND, OR, XOR, etc). Las expresiones son evaluadas por el ensamblador en tiempo de compilación y el valor obtenido es lo que se pone como argumento en el programa objeto.

Cada sentencia de un programa en assembler consta de varios campos:

[ETIQUETA] [INSTRUCCION O SEUDOINSTRUCCION] [COMENTARIO]

Todos los campos son opcionales y eso se indica usualmente con los paréntesis rectos. Esto significa que incluso una línea vacía es una sentencia válida. Los separadores utilizados para distinguir los diferentes campos dependen de cada ensamblador. Lo más usual es utilizar los dos puntos : para identificar el final de la etiqueta y un punto y coma ; para señalar el inicio del campo de comentarios.

Cuando aparece, el campo ETIQUETA contiene un identificador. Ese identificador designa al valor del contador de posiciones $ antes de la sentencia. En otra parte del programa podemos utilizar ese identificador para referirnos a la dirección asociada a $ en ese momento.

El campo de COMENTARIOS se utiliza a efectos de clarificar el contenido de un programa y es ignorado por el ensamblador. Es recomendable incluir comentarios prácticamente en todas las líneas del programa e incluso agregar líneas exclusivamente de comentarios describiendo por ejemplo la utilización de los registros del microprocesador. El uso adecuado de los comentarios facilita la comprensión de nuestro programa por otros programadores o incluso por nosotros mismos cuando más adelante lo retomamos para corregirlo o modificarlo.

El campo central puede contener o bien una INSTRUCCION o bien una DIRECTIVA para el ensamblador.

Una instrucción consta de un MNEMONICO del repertorio del microprocesador y, cuando corresponda, uno o más operandos. Por ejemplo:

```
LD B, 05H
```

Al ensamblar esa línea se deberá obtener el código de operación de esa instrucción (00000110 para el ejemplo) y los operandos (00000101). Además se incrementa el contador de posiciones en la cantidad de bytes que ocupa la instrucción (2 en este caso).

Las directivas o seudoinstrucciones soportadas varían de un ensamblador a otro, así como las reglas de sintaxis para escribirlas. Se enumeran a continuación las más importantes que aparecen en la mayoría de los ensambladores.

* <u>Origen</u>

Sintaxis:

```
ORG exp
```

Se utiliza para fijar la dirección de inicio en memoria de un segmento de programa. Para ello el ensamblador inicializa el contenido del contador de posiciones con el valor obtenido de evaluar exp. Si queremos ensamblar un programa que comience en la dirección FF hexadecimal deberemos incluir al inicio del programa fuente la sentencia siguiente:

```
ORG  0FFH
```

* <u>EQU</u>

Sintaxis:

```
nombre    EQU  exp
```

Se hace corresponder el valor de la expresión **exp** al identificador **nombre**. **nombre** puede ser utilizado en otras expresiones y será sustituido por el ensamblador. El identificador **nombre** no puede estar definido previamente, de lo contrario el ensamblador reportará un error.

```
PEPE EQU 0FH
```

En otra parte del programa puedo escribir:

```
LD A, PEPE
```

* <u>Directivas de uso de memoria</u>

```
[etiq:]   DB   [valor_inicial] ;Define Byte
          DEFB
```

```
[etiq:]   DW   [valor_inicial] ;Define Word
          DEFW

[etiq:]   DS exp, [valor_inicial] ;Define Space
```

Sirven para reservar espacio en memoria. Con la sentencia DB se reserva un byte. Se incrementa el contador de posiciones en 1. Si se incluye el identificador **etiq** se le asigna el valor del contador de posiciones anterior al ensamblado de la sentencia que es la dirección del lugar de memoria reservado. Si se incluye la expresión **valor_inicial** el ensamblador inicializa el lugar reservado.

Las directivas DW y DS son similares salvo que en DW el lugar reservado son 2 bytes (llamado a veces palabra o word) y en DS la cantidad de bytes reservada esta dada por el valor de la expresión **exp**.

p. ej.:

```
LETRA1:   DB   'A'
BUFFER:   DS   80
```

Reserva 1 byte que se inicializa con el código ascii correspondiente al carácter A y se reservan 80 bytes que no se inicializan.

5. Pines y señales del Z80.

El Z80 es un chip DIP (Dual In-line Package) de 40 patas. Se denominan DIP todos los chips cuyas patas están ordenadas como dos filas paralelas.

En la figura se representa un Z80 con todas sus patas ordenadas de acuerdo a su función.

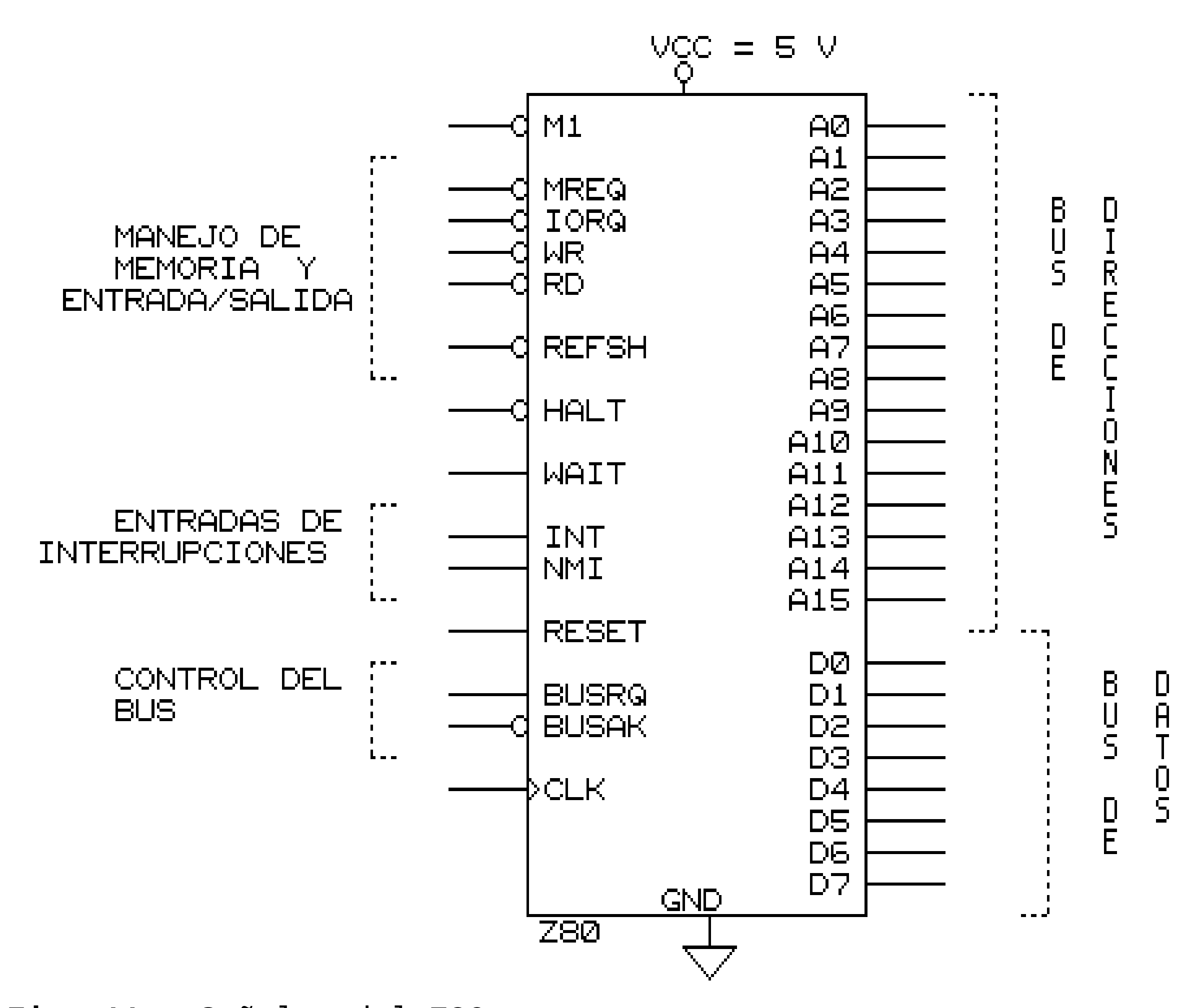


Fig. 11 - Señales del Z80.

5.a. BUS DE DIRECCIONES Y DATOS

El bus de direcciones está representado por las señales A15 a A0, siendo A15 el bit más significativo del bus de direcciones. Tienen salida en tercer estado. Esto significa que cuando el bus de direcciones está inactivo, sus salidas están en estado de alta impedancia. Como ya vimos, al poderse generar 2^{16} direcciones distintas, el tamaño máximo de la memoria externa que se puede direccionar directamente es de 65536_{10} bytes, o 64 Kbytes.

Al direccionar dispositivos de I/O, solo se utilizan las 8 líneas menos significativas del bus de direcciones, A7..A0. Por lo

tanto, el espacio de direccionamiento de entrada salida es de 2^8 = 256 bytes.

También se pone en las líneas A7..A0, en determinado momento[5], el contenido del registro de refresco de memoria, R.

El bus de datos, compuesto por las ocho líneas D7..D0, siendo D7 el bit más significativo, es bidireccional y con salida en tercer estado.

5.b. SEÑALES DE MANEJO DE MEMORIA Y ENTRADA-SALIDA.

A los efectos de determinar si el contenido del bus de direcciones debe ser interpretado como una dirección de memoria o de E/S, existen dos señales,[6] MREQ\ (requerimiento de memoria) e IORQ\ (requerimiento de E/S). Se trata de dos salidas tri-state, activas por nivel bajo. Es obvio que solo una de las dos puede estar activa en un instante dado. Estas señales participarán en la generación de la señal de habilitación de un chip de memoria o un puerto de entrada-salida.

Entretanto, las señales RD\(escritura) y WR\(lectura) indican la dirección en que habrá de efectuarse la transferencia de datos. RD\ implica una transferencia desde la memoria (o el puerto de E/S) hacia el microprocesador, en tanto WR\ indica lo contrario. Nuevamente, es obvio que no pueden estar ambas activas simultáneamente. Son señales tri-state, activas por nivel bajo.

Pertenece también a este grupo la señal de refresco de memoria, RFSH\. Un refresco de memoria RAM dinámica consiste esencialmente en una lectura de la memoria, sin que los datos sean transferidos hacia el microprocesador. Sin entrar a explicar el funcionamiento de la memoria dinámica, digamos que mientras las señales RFSH\ y MREQ\ están activas, se pone en el bus de direcciones el contenido del registro R, y que esto se utiliza para realizar el refresco de la memoria dinámica, si es que existe.

5.c. LINEAS DE CONTROL DEL BUS.

Asociadas con los buses de control, direcciones y datos, existen dos señales, la entrada activa por nivel bajo BUSRQ\ (Requerimiento de bus) y la salida activa por nivel bajo BUSAK\ (reconocimiento de bus).

Si un dispositivo externo desea poder manejar él mismo (sin la intervención del Z80) los buses del sistema, solicita mediante un nivel bajo en BUSRQ\. El Z80 responde entonces colocando los buses de direcciones, datos y salidas de control en tercer estado (alta impedancia) y bajando BUSAK\, para indicar al dispositivo externo que puede tomar el control del bus. Esta función se utiliza, por

---

[5] - Durante el ciclo M1, que veremos después.

[6] En este trabajo, para indicar señales activas por nivel bajo se utiliza la notación SEÑAL\ en lugar de la habitual barra sobre el nombre de la misma.

ejemplo, para realizar acceso directo a memoria (DMA), lo cual está más allá del alcance del presente trabajo.

5.d. OTRAS SEÑALES

La señal M1\ es activa por nivel bajo, e indica que el procesador está en un ciclo de búsqueda de una instrucción.

La señal de RESET\ también es activa por nivel bajo. Esta línea debe llevarse a nivel bajo inmediatamente después de la conexión de la alimentación al sistema, o en cualquier ocasión en que se quiera resetear el sistema.

Al llevar RESET\ a nivel bajo, ocurren las siguientes acciones[7]:

1. Se pone en el contador de programa (PC) el valor 0000H.
2. El flip-flop de habilitación de interrupciones es borrado, deshabilitando las interrupciones excepto la NMI.
3. El registro I (Registro de Vector de Interrupción) se carga con el valor 00H.
4. El registro R (Registro para refresco de memoria) se pone en 00H.
5. Se selecciona el modo 0 de interrupciones.
6. El bus de direcciones se pone en estado de alta impedancia.
7. El bus de datos se pone en estado de alta impedancia.
8. Todas las señales de control van a su estado inactivo.

RESET\ debe estar activa durante al menos 3 ciclos de reloj para garantizar que se complete toda la inicialización. Luego que RESET\ vuelve a estado inactivo (alto), transcurren dos ciclos de reloj hasta que la CPU reasuma su funcionamiento normal. Dicho de otro modo, dos ciclos de reloj después que sube RESET\, el Z80 realiza la búsqueda del código de operación que se encuentra en la posición de memoria 0000H.

La señal WAIT\ está asociada con memorias o dispositivos de E/S lentos. Mientras esta pata (que es una entrada al Z80, activa por nivel bajo) sea mantenida en estado activo, el Z80 "hará tiempo", esperando, para que la memoria o el dispositivo de E/S pueda efectuar la transferencia de datos. En resumen, esta habilidad permite que memorias o dispositivos de E/S lentos sean conectados al microprocesador. Al estudiar los diagramas de tiempos, este concepto quedará mucho más claro.

La señal HALT\ es una salida que se pone activa (baja) cuando se ha ejecutado una instrucción homónima, ya explicada.

5.e. ENTRADAS DE INTERRUPCIONES

La descripción de estas señales la postergamos hasta tratar el

[7] - Si bien se describen acciones referidas a interrupciones y otros temas que aún no han sido expuestos, se incluyen aquí a los efectos de tener un panorama claro de todas las acciones que se llevan a cabo.

tema de interrupciones (capítulo 9).

6. Ciclos de máquina

Durante la operación normal, el microprocesador busca secuencialmente y ejecuta una instrucción después de otra. La búsqueda y ejecución de todas las instrucciones pueden dividirse en un conjunto de ciclos básicos. En primer lugar, tenemos el ciclo de reloj, o ciclo T. Si se utiliza un reloj de 4 MHz, la duración del ciclo T será de 250 nanosegundos. Los ciclos T se agrupan en ciclos de máquina, o ciclos M. Cada ciclo de máquina corresponde a un acceso a memoria o a E/S. Cada instrucción ejecutada por el Z80 consta de entre uno y seis ciclos de máquina (exceptuando las instrucciones que trabajan con bloques).

En el caso del Z80, existen 7 tipos de ciclos de máquina:

1. Búsqueda de código de operación ("OPCODE FETCH"). Este ciclo se llama **M1**.
2. Lectura o escritura de memoria.
3. Lectura o escritura de E/S.
4. Requerimiento/concesión del bus.
5. Solicitud/Reconocimiento de interrupción.
6. Solicitud/Reconocimiento de interrupción no enmascarable.
7. Salida de un HALT.

Describiremos aquí sólo los ciclos de máquina más importantes. La especificación completa de todos ellos, así como las cotas de tiempo asociadas, se encuentran en la hoja de datos del microprocesador [6].

6.a. CICLO M1 - OPCODE FETCH

Previo a la ejecución de toda instrucción, el microprocesador debe leer su código de operación de memoria. Este es, generalmente, de 1 byte de longitud, pero existen instrucciones que tienen dos bytes de opcode. Por cada byte de opcode, el microprocesador realiza un ciclo M1[8].

Durante un ciclo M1, la CPU lee el opcode, lo decodifica, y ejecuta una parte (en casos de instrucciones muy sencillas, toda) de la instrucción.

Un ciclo M1 consta de 4 ciclos T, como muestra la Fig. 12 .

Al comenzar el ciclo[9], la señal M1 baja, para indicar que se trata de un ciclo de este tipo. Se pone el contenido del contador de programa en el bus de direcciones, a los efectos de realizar la búsqueda del opcode. En el flanco descendiente de T1 bajan las

[8] - Notar que no se habla aquí del tamaño total de la instrucción, sino solamente de su código de operación. Como ya se mencionó, existen instrucciones de hasta 4 bytes.

[9] - Aquí se describen los ciclos en forma cualitativa. Los tiempos exactos deben buscarse en la hoja de datos del Z80.

señales MREQ\ y RD\, indicando a la memoria externa que hay una dirección válida en el bus de direcciones.

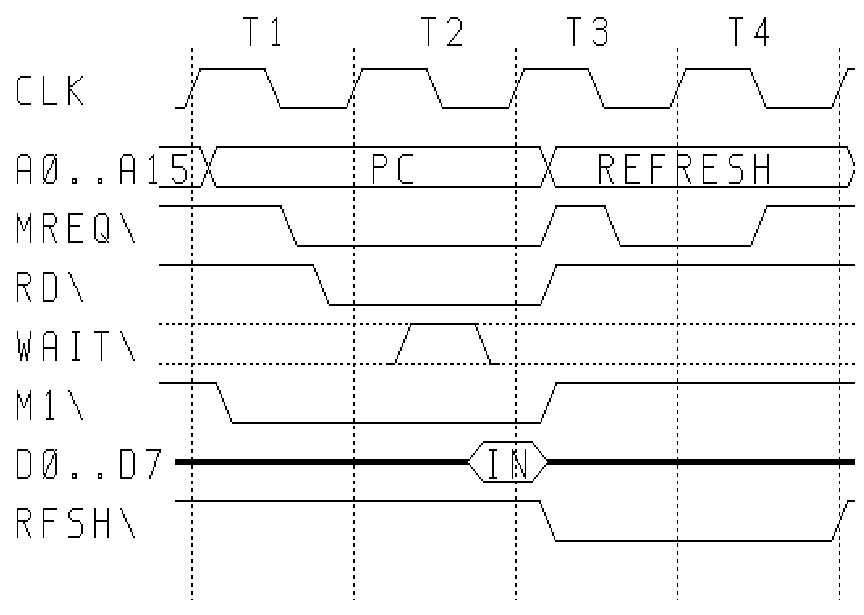


Fig. 12 - Ciclo M1

La memoria externa pondrá los datos contenidos en la dirección apuntada por el PC en algún instante previo al flanco de subida de T3 (salvo que sea una memoria lenta y requiera estados de espera). En el flanco de subida de T3, el opcode es leído por el microprocesador. Enseguida, RD\, MREQ\ y M1\ van a estado inactivo.

Los dos ciclos que restan se utilizan para proveer tiempo de refresco a la memoria dinámica externa. Por ello, bajan RFSH\ y MREQ\. El bus de direcciones (en realidad, solo las líneas A6 .. A0) tendrá ahora el contenido del registro R, que contiene la dirección de refresco.

Durante los últimos dos ciclos T, la CPU decodifica la instrucción y en casos en que no se requiere ningún acceso a memoria para ejecutarla, lo hace inmediatamente, sin necesidad de nuevos ciclos de máquina.

6.b. CICLO DE LECTURA-ESCRITURA DE MEMORIA.

Consideremos que se ha de ejecutar una instrucción LD A,(HL) que carga el acumulador con el byte apuntado por el par de registros HL. Primero, la CPU ejecuta un ciclo M1, busca y decodifica el opcode. Luego, debe realizar una lectura de memoria, a los efectos de traer el dato al acumulador.

El bus de direcciones, MREQ\ y RD\ se activan igual que en el caso de un ciclo M1. En este caso, se pondrá en el bus de direcciones el contenido de HL. La memoria pone sus datos sobre el bus de datos, y en el flanco decreciente de T3, la CPU lee el dato, cargando el registro A.

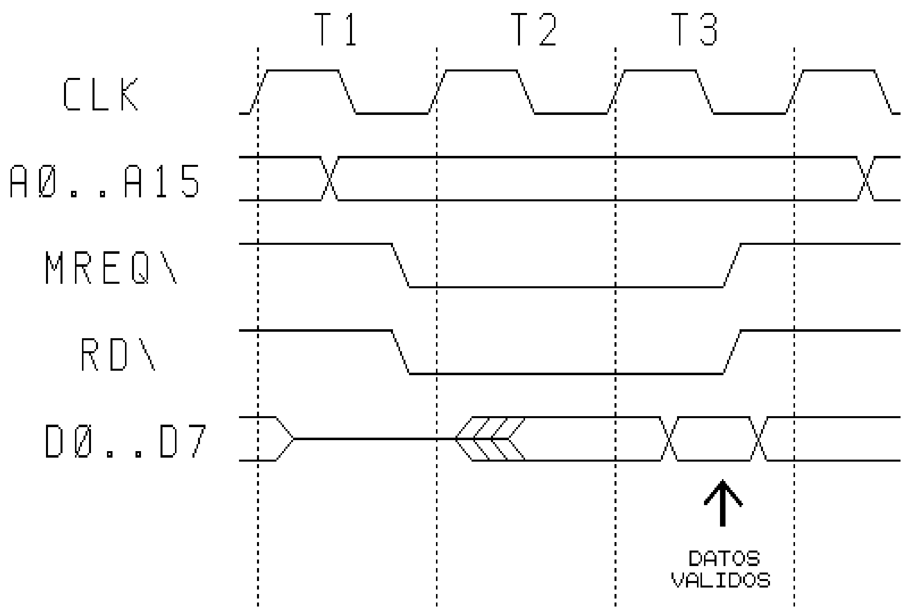


Fig. 13 - Ciclo de lectura.

En un ciclo de escritura, la señal MREQ\ y el bus de direcciones se comportan en forma idéntica. La CPU pone los datos en el bus luego del flanco de caída de T1, y los mantiene hasta el final del ciclo. Por otra parte, baja WR\ en el flanco de bajada de T2.

6.c. CICLOS DE LECTURA-ESCRITURA DE E/S

Ocurren cuando se ejecuta una instrucción IN o OUT, o sus versiones para bloques de datos: INIR, INDR, OTIR, OTDR.

Como puede verse en la hoja de datos del microprocesador, la diferencia fundamental con los ciclos de acceso a memoria son:

- En lugar de MREQ\, aquí se activa IORQ\.
- Solo el byte bajo del bus de direcciones se activa.
- Este ciclo de máquina dura 4 ciclos de reloj, debido a que el microprocesador inserta siempre (es decir, sin que se lo pidan) un ciclo de espera.

6.d. LOS CICLOS DE ESPERA[10].

En general, los ciclos de espera (frecuentemente llamados "wait states") pueden solicitarse durante cualquier acceso a memoria o E/S. Cuando la memoria o el dispositivo de E/S recibe una señal RD\ o WR\ junto con MREQ\ o IORQ\, pueden responder enviándole una señal WAIT\ a la CPU. La CPU detectará esta condición, difiriendo la continuación del ciclo de bus hasta que se levante la

[10] - En el apartado 7.d se describe el diseño de un circuito generador de ciclos de espera

pata WAIT\. Nótese[11] que los ciclos de espera no se insertan en cualquier parte del ciclo de máquina. El microprocesador los coloca siempre entre T2 y T3, y mantiene el estado de las líneas como quedaron al final de T2. Como se puede observar, siempre al final de T2 las señales que participan en la decodificación del dispositivo (A15..A0, MREQ\, IORQ\, RD\ o WR\, según sea el caso) están activas. Por esa razón, el ciclo de espera hace que, efectivamente, el microprocesador se "enlentezca" para adaptarse a los requerimientos del periférico.

6.e. EL ESTADO DEL BUS

En muchos casos es importante determinar por hardware, qué tipo de ciclo de máquina está realizando la CPU. Por ejemplo, considérese la situación en que debe insertarse un ciclo de espera solo en los ciclos M1. También se presentan a menudo la necesidad de distinguir un ciclo de reconocimiento de interrupciones.

A los efectos de clarificar este tema, presentamos en la siguiente tabla las señales que identifican cada ciclo de máquina.

CICLO	MREQ\	IORQ\	RD\	WR\	M1\
M1[12]	0	1	0	1	0
LECTURA MEM.	0	1	0	1	1
ESCRITURA MEM.	0	1	1	0	1
LECTURA E/S	1	0	0	1	1
ESCRITURA E/S	1	0	1	0	1
RECONOC. INT.	1	0	1	1	0

[11] Ver hoja de datos.

[12] - Se refiere a la primera parte del ciclo M1, cuando lee el código de operación.

7. Sistema básico basado en Z80

7.a. LO IMPRESCINDIBLE

Los componentes básicos requeridos para un sistema mínimo basado en un Z80 son:

1. Una fuente de 5 volts.
2. Un circuito que genere la onda de reloj.
3. Un mecanismo para resetear el sistema.
4. Una ROM que contenga el programa.
5. Dispositivos e interfases de E/S.
6. El Z80.

La Fig. 14 muestra un sistema muy sencillo basado en el microprocesador Z80, que contiene todos los elementos básicos.

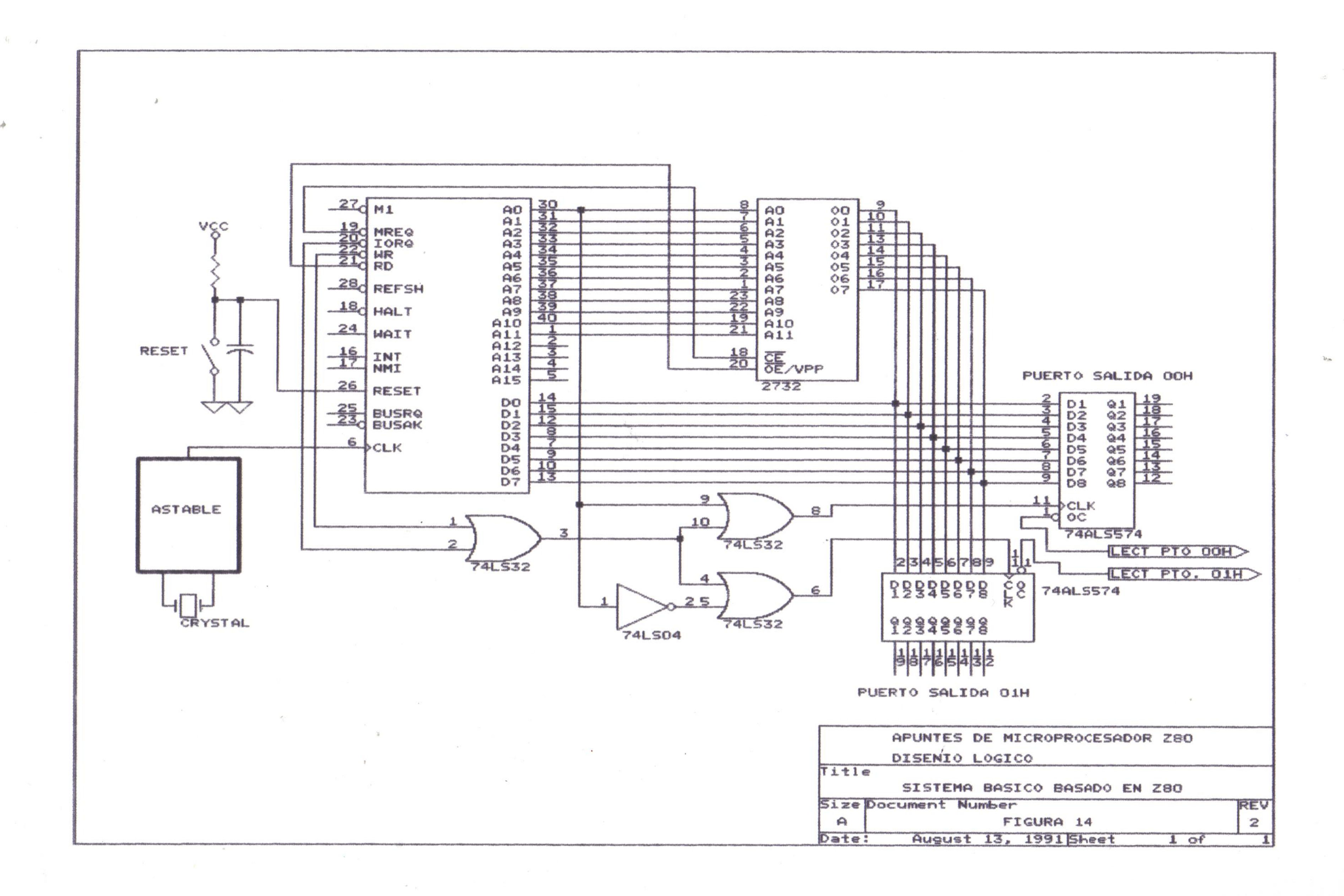
El cierre de la llave de reset provoca la descarga del condensador ubicado entre la pata de RESET\ y tierra, poniéndose un cero en ese punto. Al abrir la llave, se carga el condensador a través de la resistencia. R y C se calcularán de modo que el tiempo mínimo que permanece RESET\ en cero sea mayor que el mínimo necesario.

El "astable" será un circuito generalmente controlado con un cristal (a los efectos de fijar con precisión la frecuencia), que da una onda rectangular con un ciclo de trabajo y una frecuencia determinados. Funciona en forma continua, mientras esté alimentado el sistema.

Hemos colocado una memoria EPROM de 4K X 8, por lo cual necesitamos 12 líneas del bus de direcciones para seleccionar una palabra. Se eligieron las líneas más bajas, A11..A0, y se dejó sin conectar las cuatro más altas A15..A12. Por esta razón, todas las direcciones que finalicen con los mismos 12 bits harán referencia a una misma posición de memoria. Debe destacarse que se puede utilizar esta solución (la de ignorar las líneas de direcciones que no van directamente conectadas al chip) debido a que existe solo un chip de memoria. Si existieran varios, deberían utilizarse las líneas A15..A12 para seleccionar uno de ellos.

En principio, no está previsto que esta memoria requiera ciclos de espera, pero ello quedará determinado recién cuando hagamos el estudio de tiempos correspondiente.

Veamos a continuación cómo se han conectado las patas de control de la EPROM. Consideremos que esta memoria contendrá el programa, el cual se accederá realizando ciclos M1 o ciclos de lectura de memoria. Ambos se caracterizan por la bajada de las señales MREQ\ y RD\. La primera de ellas, significa que se realizará un acceso a memoria, y por lo tanto, se la utiliza para seleccionar la memoria (CE\ = chip enable). Al bajar CE\, se habilitan los decodificadores internos de la EPROM, a los efectos de determinar qué byte debe ponerse a la salida, de acuerdo con la



VCC
RESET
ASTABLE
CRYSTAL
M1
MREQ
IORQ
WR
RD
REFSH
HALT
WAIT
INT
NMI
RESET
BUSRQ
BUSAK
CLK
74LS32
74LS04
2732
CE
OE/VPP
PUERTO SALIDA 00H
74ALS574
LECT PTO 00H
LECT PTO. 01H
PUERTO SALIDA 01H
APUNTES DE MICROPROCESADOR Z80
DISEÑIO LOGICO
Title
SISTEMA BASICO BASADO EN Z80
Size
A
Document Number
FIGURA 14
REV
2
Date: August 13, 1991
Sheet 1 of 1

dirección de entrada. Sin embargo, la EPROM no pone los datos sobre el bus de datos hasta que no se active su otra entrada de control, OE\ (= output enable). Quiere decir que ésta última se debe bajar cuando el microprocesador está en condiciones de leer: la manejamos con la pata RD\.

Observando los diagramas de tiempos, se nota que el comportamiento de MREQ\ y RD\ en los dos ciclos de máquina mencionados es prácticamente idéntico. Por lo tanto, la distinción que hicimos previamente sólo es importante desde el punto de vista conceptual, en el caso del Z80. Dicho de otra forma, muy probablemente un sistema basado en este microprocesador funcione tan bien con las conexiones RD\ -- OE\ y MREQ\ -- CE\, como con las conexiones cruzadas, es decir, RD\ -- CE\ y MREQ\ - OE\. Lo importante es que esta última conexión no funcionará para muchos procesadores.

Otra característica del sistema de la figura es que la única memoria de que se dispone a los efectos de guardar variables son los 14 registros de la CPU. Asimismo, no se puede implementar el stack, ya que para ello se requiere que el sistema posea RAM externa. Esto significa que el programa no podrá contener las instrucciones PUSH, POP, CALL, RET, y todas aquellas que requieran una escritura de memoria.

Se ha provisto al sistema de dos puertos de salida, implementados mediante latches. Como estos latches solo deben ser escritos por operaciones de E/S, se los selecciona utilizando IORQ\ y WR\ (es decir, escritura de E/S). Para distinguir uno de otro, se utiliza A0: toda escritura de E/S cuyo destino sea una dirección par (es decir, con el LSB = 0) escribirá sobre el latch indicado como "puerto salida 00H" en la Fig. 14 y análogamente con las escrituras a direcciones impares. Las salidas de estos latches se activan poniendo en 0 la pata OC\. Dado que son puertos de salida, estas patas son manejadas por el dispositivo externo, no por el microprocesador.

7.b. DECODIFICACION DE MEMORIA: UN EJEMPLO

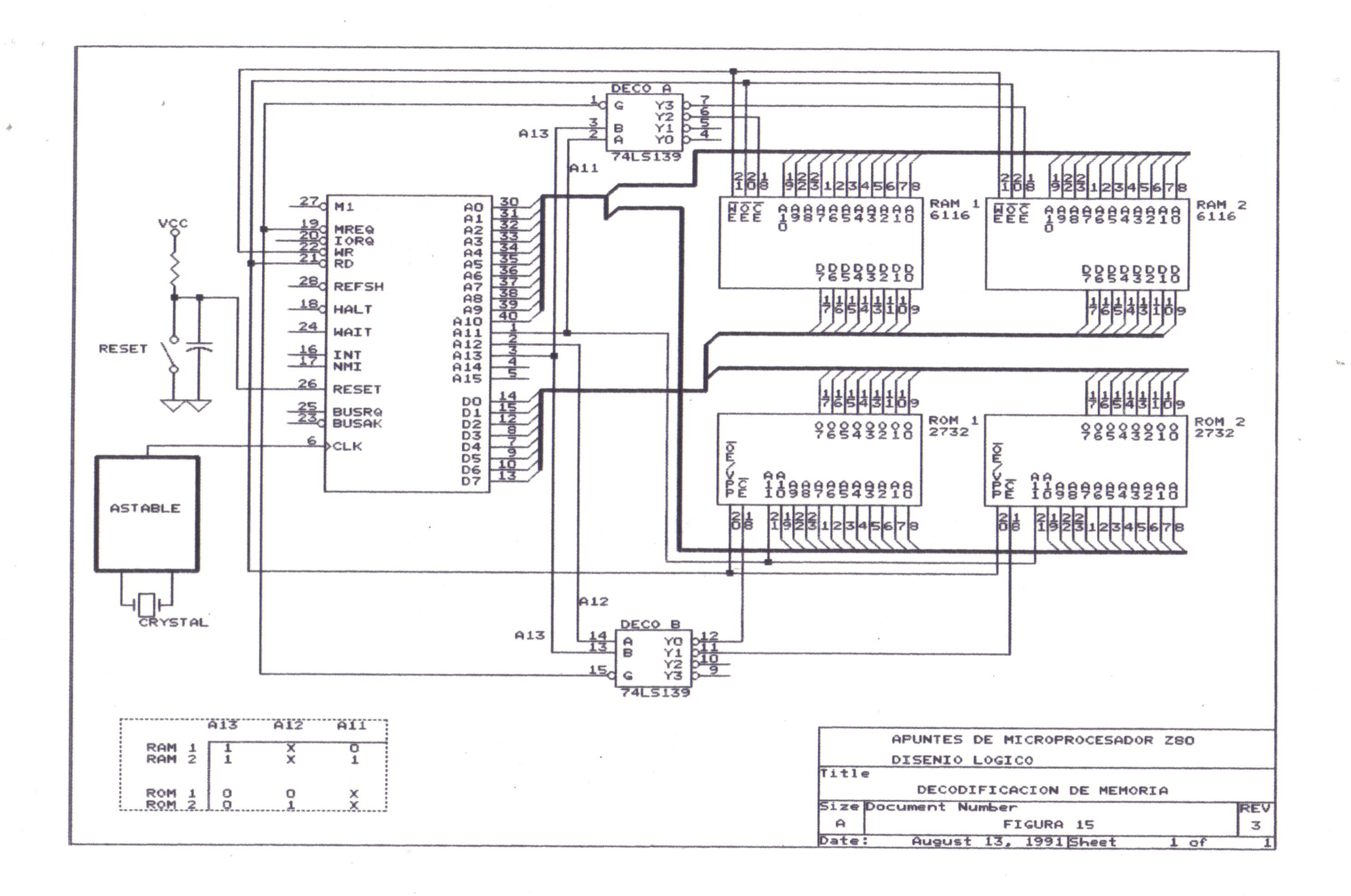
Utilizaremos el esquema de la Fig. 15 ("decodificación de memoria") para dar un ejemplo de un sistema que contiene ROM y RAM.

En primer lugar, observemos que en lo referente a reset y reloj el sistema es idéntico al anterior. También podría haberse incluido la parte de E/S, que se omitió exclusivamente por razones de simplicidad del esquema.

El sistema consta de dos chips de EPROM 2732 (4 Kbytes cada uno) y dos chips de RAM 6116 (2 Kbytes cada uno). Se eligió seleccionar con la línea A13 el tipo de memoria:

* Si A13 = 0, se está accediendo a memoria ROM
* Si A13 = 1, se accede a RAM.

Obsérvese que no es correcto hacer la otra elección posible, ya que luego de un reset el sistema comenzará a ejecutar en la dirección 0000H, es decir, tendrá A13 = 0, y se pretende que allí

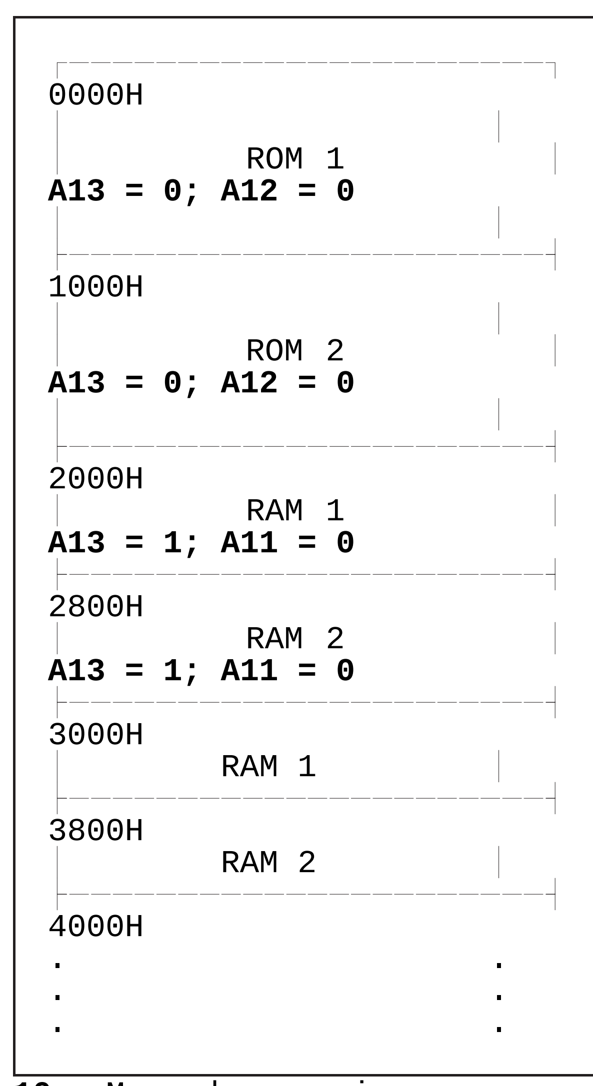


	A13	A12	A11
RAM 1	1	X	0
RAM 2	1	X	1
ROM 1	0	0	X
ROM 2	0	1	X

exista memoria no volátil.

En el caso de la RAM, se utilizan A10..A0 para la decodificación interna, y A11 para elegir entre el chip "RAM 1" o "RAM 2".

Fig. 16 - Una alternativa para el circuito de decodificación de la RAM.



10 - Mapa de memoria correspondiente al circuito.

Se podría haber utilizado la lógica de la Fig. 16 para lograr esta decodificación, pero se resolvió implementarla con un decodificador 74LS139. La razón de esta elección es que en un solo chip se tienen dos de estos decodificadores, y se elimina la necesidad de utilizar otros chips que contengan compuertas para implementar el sistema. En cuanto a costo, no es mucho más caro un chip con decodificadores que uno con compuertas.

En el recuadro 10 se observa el mapa de memoria resultante. Se muestran solo los primeros 16 Kbytes, ya que luego se repite. Observar que como se ignora A12 al decodificar la RAM, esta se mapea repetida.

7.c. ESTUDIO DE TIEMPOS: UN EJEMPLO.

A los efectos de ejemplificar el estudio para determinar la necesidad de introducir ciclos de espera, lo haremos para el caso de la interconexión entre la EPROM 2732 y el microprocesador, en el caso del sistema básico (Fig. 14), donde no existen compuertas ni decodificadores intermedios[13].

Trabajaremos con los tiempos del microprocesador que aparecen en la hoja de datos, considerando un reloj de 4 MHz y ciclo de trabajo 50% (es decir, onda cuadrada). Los tiempos de subida y bajada de esta onda cuadrada se considerarán despreciables.

[13]- En otro caso, debería considerarse los retardos introducidos por la lógica adicional.

En cuanto a los tiempos asociados a la EPROM, tomaremos los siguientes:

- t_{ACC} = (Address to output delay) = 450 nseg max.
- t_{CE} = (CE\ to output delay) = 450 nseg max.
- t_{OE} = (OE\ to output delay) = 150 nseg max.
- t_{DF} = (OE\ high to output float) =
  0 min.
  130 max.

Las inecuaciones que se obtienen al estudiar un ciclo M1 son:

$$t_{su} = T1 - t_6^{max} + T2 + nTW - t_{ACC} > t_{15}^{max} \qquad (1)$$

donde hemos empleado la siguiente notación:

- t_{su} = tiempo de setup que podemos garantizar.
- Ti = el ciclo i de reloj (= 250 nseg).
- TW = ciclo de espera.
- n = número de ciclos de espera (es la incógnita).
- t_i = tiempo i-ésimo de la hoja de datos del Z80, por ejemplo, t_6 = TdCr(A).

La condición (1) tiene el siguiente significado: t_{ACC} es lo máximo que demora la EPROM en presentar los datos correctos en su salida (es decir, sobre el bus de datos) a partir del instante en que se ponen las direcciones válidas. Esto ocurre t_6 nseg después del comienzo de T1. Por otra parte, el microprocesador lee los datos en el flanco creciente de T3, por lo cual el tiempo de setup será el intervalo previo a dicho flanco en que los datos están estables. Si la EPROM pusiera los datos instantáneamente en el bus de datos, el tiempo de setup que tendríamos sería lo que resta del ciclo T1 , es decir T1 - t_6, mas el ciclo T2, mas los n ciclos de espera TW. A los efectos de que el microprocesador lea correctamente estos datos, t_{su}^{min} debe satisfacer el tiempo de setup del Z80 en el peor caso, de donde sale (1).

$$t_{su} = T1^{low} - t_{13}^{max} + T2 + nTW - t_{OE} > t_{15}^{max} \qquad (2)$$

t_{OE} es lo máximo que demora la EPROM en presentar los datos correctos en su salida a partir del instante en que se activa (baja) la entrada OE\ (suponiendo que se verifica t_{ACC}).

La ecuación (2) surge, al igual que (1), de imponer que el menor setup real considerando t_{OE} sea menor que el mínimo tiempo exigible para respetar el setup del Z80.

La inecuación para t_{CE} es análoga:

$$t_{su} = T1^{low} - t_8^{max} + T2 + nTW - t_{CE} > t_{15}^{max} \qquad (3)$$

En cuanto a t_{DF}, es el tiempo que permanecen los datos luego que subió OE\, y debe satisfacer el hold time del microprocesador:

$$t_{DF}^{min} >= t_{16}^{max} \qquad (4)$$

Obsérvese que de no cumplirse esta inecuación, ello no se

puede solucionar agregando estados de espera.

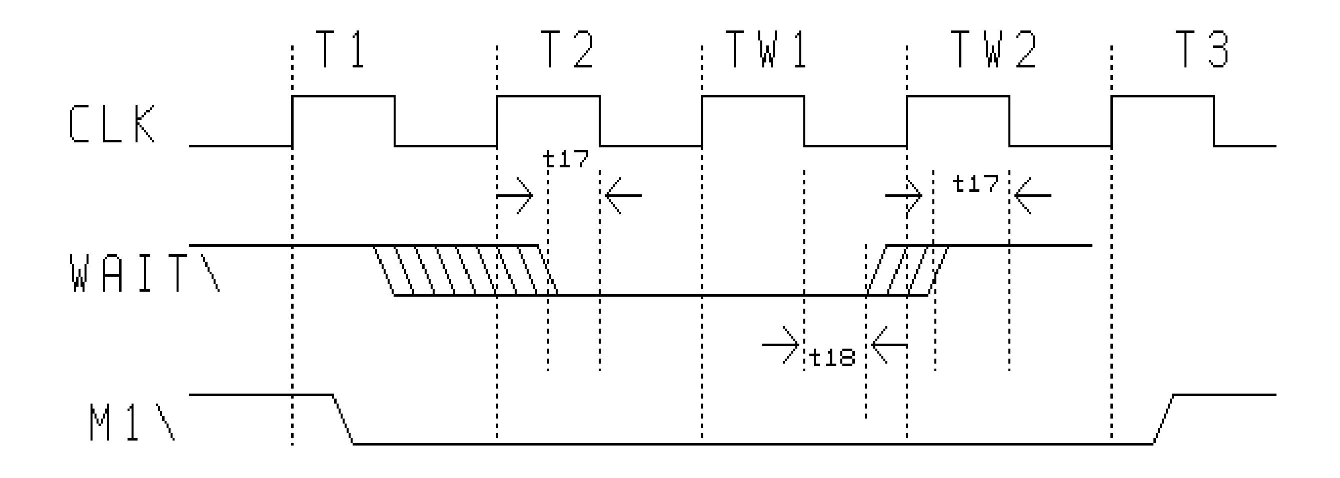
```
250 - 110 + 250 + 250n - 450 > 35       (1)
125 -  95 + 250 + 250n - 150 > 35       (2)
125 -  85 + 250 + 250n - 450 > 35       (3)
0 >= 0                                  (4)

95  < 250n ==> n >= 1         (1)
-95  < 250n ==> n >= 0        (2)
195 < 250n ==> n >= 1         (3)
```

Para que se satisfagan las tres inecuaciones necesitamos introducir 1 TW.

A los efectos de completar este análisis, debería estudiarse el ciclo de lectura de memoria EPROM, los ciclos M1[14], de lectura y escritura de la RAM, y los ciclos de escritura de E/S de los latches.

7.d. UN GENERADOR DE CICLOS DE ESPERA.

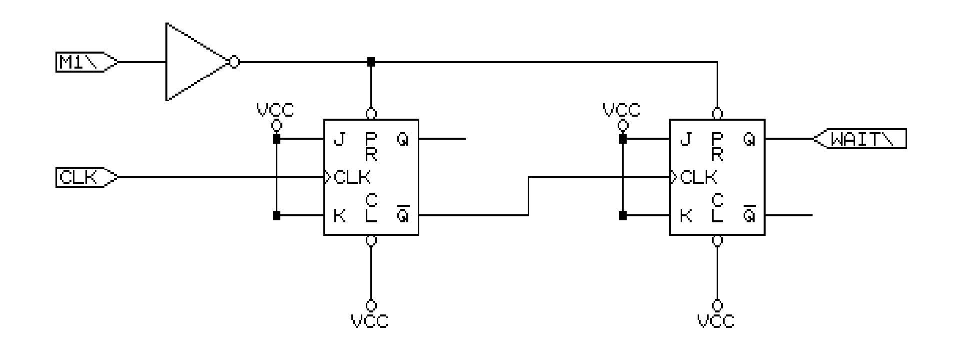


En la Fig. 17 se ven las especificaciones que debe cumplir la señal WAIT\ a los efectos de que se produzcan ciclos de espera. En primer lugar, WAIT\ debe bajar t_{17} nanosegundos antes que el flanco de bajada de T2, a los efectos que el ciclo que sigue a T2 sea un TW_1. WAIT\ debe subir t_{17} nanosegundos antes de TW_n, si se desea tener solo n ciclos de espera. Es decir que t_{17} es el tiempo de setup de la entrada WAIT\ con respecto al flanco de bajada de CLK. Existe también un tiempo de hold a respetar, t_{18}.

En el circuito de la Fig. 18 se presenta un circuito que genera 2 ciclos de espera cada vez que el Z80 ejecuta un ciclo M1. En efecto, mientras M1\ esté en 1, las salidas Q de los flip flops estarán en 1. Al bajar M1\, en el siguiente flanco creciente de reloj, subirá Q1\ (salida del primer flip flop) lo que provocará un

---

[14] - En el caso que se desee ejecutar código que esté en RAM



flanco creciente en CLK2, y entonces Q2 = 0. El siguiente flanco de reloj (en TW1) llevará Q1\ a 0, y el siguiente (en TW2) lo llevará a uno, provocando otro flanco creciente en CLK2, lo cual pone Q2 en 1. Como consecuencia de esto último, el siguiente ciclo que se ejecuta es T3.

Como puede observar, un generador de n ciclos de espera es un circuito secuencial en modo reloj con una entrada (en este caso M1\, pero también podría ser MREQ\, o MREQ\ or RD\, etc) y una salida: WAIT\

8. Entrada - Salida controlada por programa

8.a. INTRODUCCION

Cualquier aplicación de un sistema con microprocesador requiere la transferencia de datos entre la circuitería externa al microprocesador y el microprocesador en sí mismo. Esta transferencia de datos existe además de las transferencias entre el microprocesador y la memoria principal y se denomina entrada/salida.

Hay varias maneras en las que se inicia y controla la transferencia de información; no obstante, pueden clasificarse en tres categorías primarias:

1. E/S controlada por programa
2. E/S controlada por interrupciones
3. E/S controlada por hardware

Los tres métodos difieren en el grado en que el microprocesador inicia y controla la transferencia de datos. En operaciones de E/S controladas por programa, la transferencia de datos está bajo el control completo del programa del microprocesador; es decir, una operación E/S tiene lugar sólo cuando se encuentra una instrucción de transferencia de E/S durante la ejecución del programa.

En contraste, si se utiliza el mecanismo de las interrupciones[15], un dispositivo externo indica directamente al microprocesador su disposición para transferir datos, poniendo en estado activo una entrada de interrupciones del microprocesador. Cuando se interrumpe al microprocesador, se suspende el programa que está siendo ejecutado y el control se transfiere a una rutina de servicio de interrupción. Esta realiza la transferencia de datos y devuelve el control al programa en el punto en que fue interrumpido. Vemos que en este caso *la transferencia se inicia mediante un hardware externo y se controla mediante la rutina de servicio de interrupción.*

Las transferencias controladas por hardware, denominadas comunmente *acceso directo a memoria, DMA,* son transferencias directas entre el dispositivo de E/S y la memoria: el dato no se dirige del dispositivo a uno de los registros del microprocesador y luego a la memoria principal, o viceversa, sino que se transfiere directamente entre el dispositivo externo y la memoria. El DMA se utiliza primordialmente para transferir un bloque de datos a alta velocidad, y está fuera del alcance de este material.

Tal como se discutió en el capítulo 1, los puertos de entrada y salida son básicamente registros externos. Algunos microprocesa-

[15] - Las interrupciones se tratan en el capítulo 9 de este trabajo.

dores proporcionan señales de control que permiten a registros externos asociados con dispositivos de E/S ocupar un espacio de direcciones separado de la memoria principal. Este es el caso del Z80. Cuando se asignan puertos de E/S al espacio de direcciones separado, se denomina *E/S estándar o aislada.* Cuando se asignan al mismo espacio de direcciones que la memoria, se denomina *E/S mapeada en memoria*.

Las señales IORQ\ Y MREQ\ del Z80 determinan si la dirección generada durante una transferencia de datos se refiere a memoria (MREQ\ = 0) o al espacio de E/S separado (IORQ\ = 0). Cada uno de ellos será posteriormente dividido por las señales de control en un espacio de direcciones de entrada (RD\ = 0) y uno de salida (WR\ = 0).

8.b. LATCHES COMO PUERTOS DE SALIDA.

Un puerto de salida debe ser capaz de almacenar datos sacados por el microprocesador hasta que el dispositivo de salida asociado con el puerto pueda aceptar dichos datos. Por lo tanto, un puerto de salida en su forma básica es simplemente un registro o latch. Un puerto de salida de 8 bits puede ser implementado con un solo chip tal como el 74ALS574 que es un conjunto de 8 flip flops tipo D sensibles al flanco positivo, y con salidas en tercer estado o el 74ALS573 que difiere del anterior en que es sensible a nivel alto.

Dado que el pulso de selección del dispositivo es derivado de la señal de WR\, los datos serán escritos en el 74ALS574 en el instante en que ocurra el flanco creciente de WR\[16]. El buffer de salida debe estar constantemente habilitado (o ser comandado por el periférico).

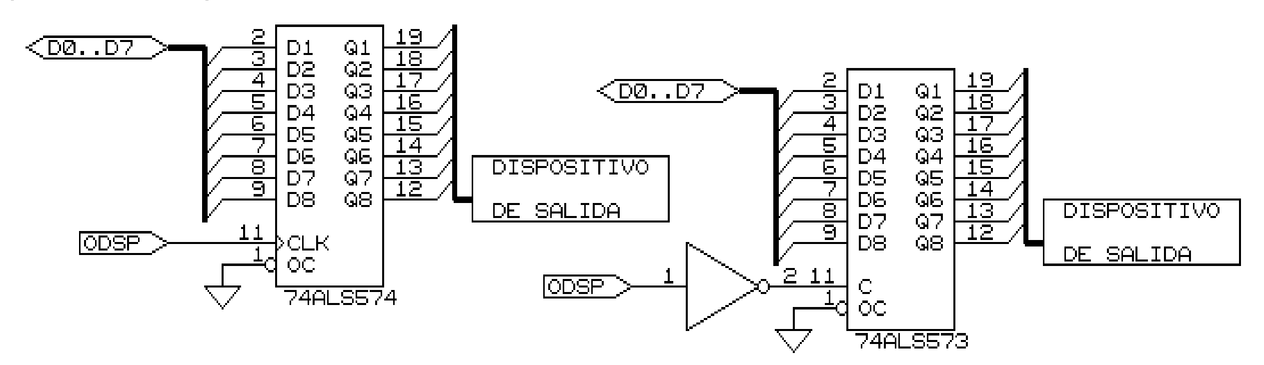


Fig. 19 - Puerto de salida básico.

En la Fig. 19 indicamos como "ODSP" el pulso de selección de dispositivo de salida ("Output Device Select Pulse"). Se supone además que los pulsos de selección son activos en nivel bajo.

Si se utiliza un 74ALS573, este se mantiene transparente

[16] - Esto no es estrictamente cierto: en realidad ocurre con el primer flanco de subida de las señales involucradas en la selección del puerto, típicamente WR\ e IORQ\.

mientras CLK está alto, es decir, se comporta como sensible al flanco de bajada. Nótese que es imperioso que el pulso de selección del dispositivo sea activo por nivel alto, ya que de lo contrario, mientras no hay pulso el latch está en estado transparente.

Es bueno resaltar que desde el punto de vista lógico estos dos registros son respectivamente idénticos al 74LS373 y 74LS374. La única diferencia es cómo están ordenadas las patas del chip: en el caso de los presentados en primer lugar están ordenadas de forma que los 8 bits de entrada están de un lado del chip, y los 8 de salida de otro, en tanto que en los últimos están todas mezcladas.

8.c. LATCHES Y BUFFERS COMO PUERTOS DE ENTRADA.

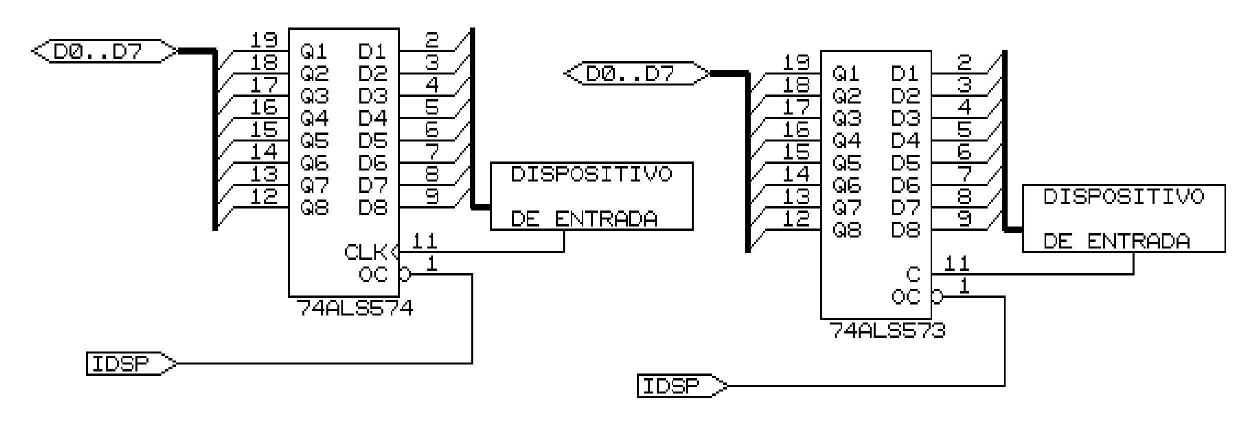


Fig. 20 - Puerto de entrada básico.

Los datos de un dispositivo de entrada deben almacenarse hasta que sean entrados por el microprocesador. El puerto de entrada debe aislar los datos del bus de datos del microprocesador hasta que éste esté listo para leerlos. El puerto de entrada más sencillo es un buffer tri-estado[17]. Esto es apropiado solo cuando el dispositivo de entrada mantiene los datos, como en el caso de que un grupo de llaves mecánicas. En caso contrario, debe utilizarse un latch cuyo reloj sea comandado por el periférico, tal como muestra la Fig. 20 .

El pulso de selección del dispositivo de entrada habilita sus buffers tri-estado. Llamaremos "IDSP" (Input Device Select Pulse) a dicho pulso.

8.d. TRANSFERENCIAS CONDICIONALES E INCONDICIONALES.

Una transferencia incondicional es aquella en la que una instrucción transporta datos de o hacia un puerto de E/S sin determinar previamente si el puerto está dispuesto para recibir o transmitir el dato. Estas transferencias manejan información de comandos, de estado, u otros datos. La información de comandos se

[17] - Notar que en el caso de un puerto de salida no puede faltar un latch, y en el caso de uno de entrada no puede faltar un buffer tri-estado.

transfiere desde el microprocesador para controlar la operación de un dispositivo de E/S. La información de <u>estado</u> se transfiere en sentido opuesto y se utiliza por el microprocesador para conocer la situación en que se encuentra el dispositivo de E/S.

Un ejemplo de transferencia de datos de salida incondicional es el comando de un display por parte del microprocesador: el microprocesador no comprueba que el display esté listo para recibir los datos, simplemente asume que lo está. La entrada de datos de un conjunto de interruptores manuales ilustra una transferencia de datos de entrada incondicional. Aquí el microprocesador asume que los interruptores están situados en sus posiciones deseadas.

En transferencias de datos condicionales, la ejecución de la instrucción de E/S que transfiere el dato está condicionada a que el dispositivo de E/S este preparado para la transferencia de datos. Dicha preparación se determina mediante una transferencia de información de estado incondicional desde el dispositivo al microprocesador. A menudo el estado de un dispositivo se indica mediante menos de 8 bits: el software simplemente ignora los bits no utilizados.

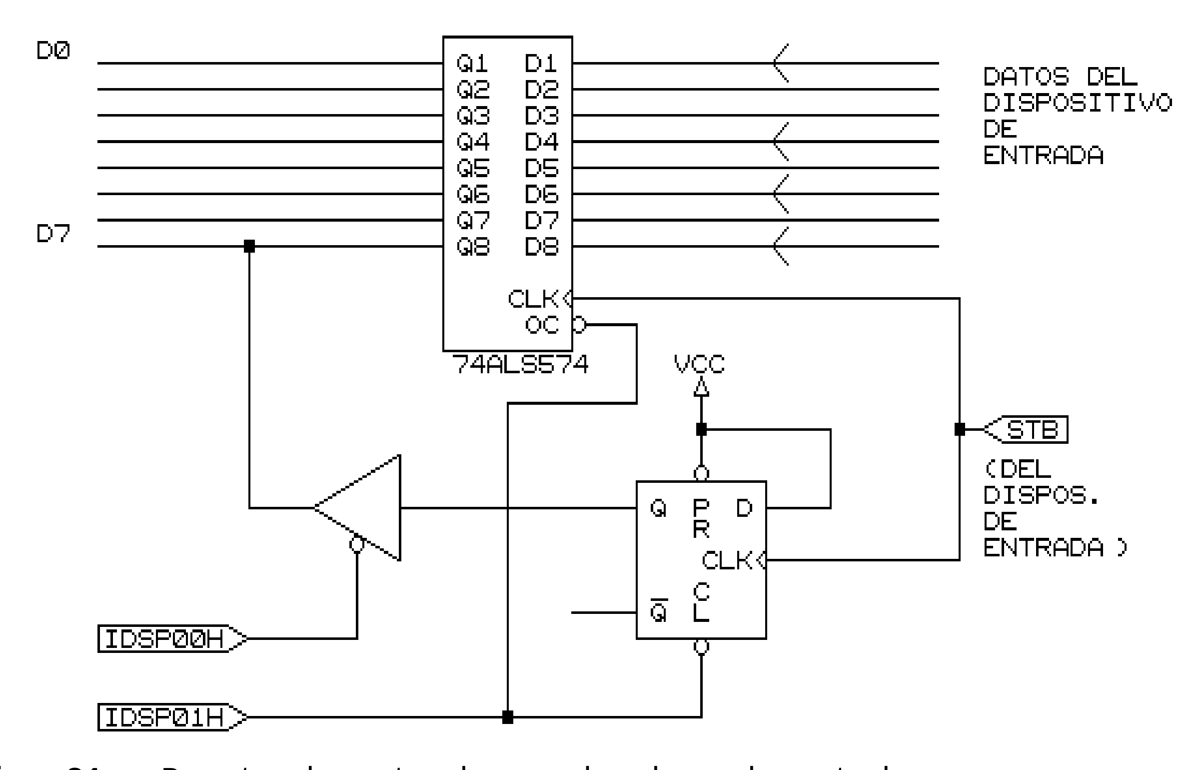


Fig. 21 - Puerto de entrada con bandera de estado.

Un solo bit de información de estado indica si un determinado puerto de entrada tiene información disponible para entrada, o cuando un dispositivo de salida está preparado para recibir información. Notemos que <u>el software que comprueba la bandera de estado incrementa el tiempo asociado con la operación de E/S</u>; el tiempo asociado es el <u>**overhead de E/S**</u>.

Considerar, por ejemplo, un dispositivo de entrada que tiene

datos disponibles en el puerto 1 (ver Fig. 21). Para indicar disponibilidad, el dispositivo de entrada pone a uno una bandera, el bit 7 del puerto de entrada 0. La utilización de flags para el control de transferencias condicionales se denomina **handshaking**. En la E/S salida controlada por programa, es la única forma de conocer si hay un nuevo dato disponible.

Para determinar la disponibilidad de datos de entrada, el microprocesador lee periódicamente la palabra de estado y comprueba el bit 7. si está en 1, hay dato disponible lo lee por el puerto 1. La señal de control que selecciona el puerto 1 para leerlo (IDSP01H) resetea la bandera de estado.

Cuando se utilizan varios dispositivos de entrada en un sistema con E/S programada, una rutina verifica por turno la bandera de cada dispositivo, para ver quien tiene datos disponibles para el microprocesador. Este proceso recibe el nombre de **polling** (en inglés, encuesta).

8.e. PUERTOS INTEGRADOS PROGRAMABLES.

Las combinaciones de latches, buffers y banderas que constituyen los puertos de E/S están disponibles como un chip. Además, un único componente puede contener varios puertos. Y para mayor versatilidad, son programables, es decir, el modo de operación de cada puerto se establece bajo el control de programa.

Son ejemplos de puertos programables el 8155A (descrito en la referencia [3]) y el Z80 PIO, que está diseñado específicamente para trabajar con el microprocesador Z80, y del que nos ocuparemos en el capítulo 10.

9. Interrupciones

9.a. INTRODUCCION

La consulta de las banderas de petición de servicio de E/S monopoliza una cantidad significativa del tiempo del microprocesador. Esto reduce el "**throughput**" del sistema (la información total útil procesada o comunicada durante un período de tiempo especificado). Por lo tanto es ventajoso, en términos de incrementar el throughput, así como para reducir la complejidad del programa, si un dispositivo de E/S pide servicio directamente al microprocesador.

Las interrupciones proporcionan esta capacidad. *Esencialmente, una interrupción es un llamada a subrutina inicializada mediante un hardware externo*. En la Fig. 22 se muestra una estructura simple que permite a un único dispositivo interrumpir al microprocesador.

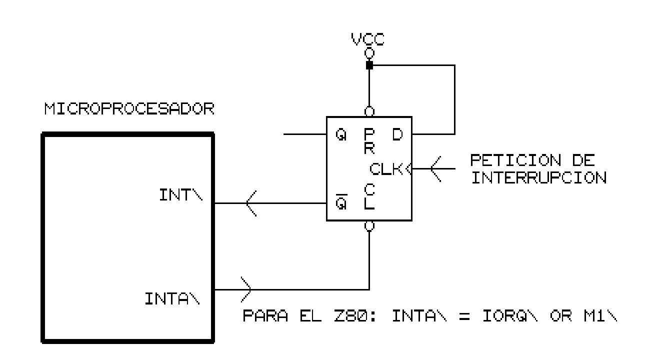


Fig. 22 - Generación de una interrupción enmascarable, en el caso de un solo dispositivo.

Cuando un dispositivo de E/S pide servicio, pone en estado activo su flip flop de petición de interrupción. Este flip flop es funcionalmente el mismo que el de petición de servicio del capítulo 8, excepto que en lugar de tener su salida conectada a un puerto de entrada, se conecta a una pata especial de interrupción del microprocesador. Así, el flip flop memoriza la petición de interrupción del dispositivo de E/S hasta que sea reconocida por el microprocesador.

Puesto que la petición de interrupción es asíncrona, puede ocurrir en cualquier punto de la ejecución de un programa. *Cuando un programa es interrumpido, la ejecución de la instrucción en curso se completa, se reconoce la interrupción por el microprocesador*[18] *y se transfiere el control a una rutina que da servicio a la interrupción*. Cuando el microprocesador responde a la interrupción, el biestable de petición de interrupción se borra mediante una señal directa del microprocesador (como en la Fig. 22) o mediante un pulso de selección de dispositivo generado por la subrutina de servicio (como el de la Fig. 21).

Para continuar la ejecución del programa en el punto adecuado cuando ha acabado la rutina de servicio de E/S, el contador de programa (PC) se guarda <u>automáticamente</u> antes de que se transfiera el control.

---

[18] - Como veremos más adelante, esto implica un ciclo de máquina especial.

La rutina de servicio de una interrupción por ser asíncrona, puede ocurrir en cualquier momento durante la ejecución del programa. Esta característica hace que -a diferencia de las rutinas comunes, que se las invoca mediante una instrucción CALL- el programa no pueda prever cuando será interrumpido y por lo tanto *es responsabilidad de quien programa una rutina de servicio de interrupción salvar el* ***estado del microprocesador****.* Ese es el nombre que se le da al conjunto de valores que contienen el contador de programa, registro de flags, acumulador y registros de propósito general en un determinado instante. Toda rutina de servicio de interrupción debe guardar el estado del microprocesador al comenzar (o por lo menos, aquellos registros que se alterarán durante la ejecución de la rutina de interrupción) y debe restaurar el estado antes de terminar. Generalmente se utiliza el stack para guardar temporalmente estos valores.

Hay dos tipos de entrada de interrupción: no enmascarable y enmascarable. Cuando se activa una entrada de interrupción no enmascarable, el microprocesador se interrumpe. Cuando se activa una entrada enmascarable de interrupción, el microprocesador es interrumpido sólo si se ha habilitado esta entrada en particular. Las interrupciones enmascarables se habilitan o inhiben bajo control de programa. Si están inhibidas, el microprocesador ignora la señal de interrupción.

En respuesta a una interrupción, ocurren las siguientes operaciones:

1. Se completa la ejecución de la instrucción en curso.

2. Se ejecuta un ciclo de máquina de reconocimiento de la interrupción y durante el mismo el contador de programa se guarda y el control se transfiere a una posición de memoria apropiada[19].

3. Al comenzar la rutina de servicio de interrupción, se guarda el estado del microprocesador (bajo responsabilidad del programador).

4. Si más de un dispositivo de E/S está asociado con la posición a la cual ha sido transferido, la rutina de servicio de interrupción identifica el dispositivo de más prioridad que pide interrupción[20].

5. Se ejecuta el resto de la rutina que da servicio al dispositivo E/S que ha interrumpido.

6. Se restaura el estado del microprocesador que ha sido guardado

---

[19] - Este paso varía según la forma en que se esté trabajando con las interrupciones. Ver 9.b.II.

[20]- Esto es válido solo en aquellos casos en que dicha identificación del dispositivo no se haya producido en el paso 2 de esta secuencia.

(también bajo responsabilidad del programador de la rutina de servicio a la interrupción). En algunos casos, se debe habilitar de vuelta las interrupciones (ver página 67).

7. Se devuelve el control a la instrucción que sigue a la instrucción interrumpida, ejecutando una instrucción especial, que en el caso del Z80 es RETI o RETN.

Cada paso requiere cierta cantidad de tiempo. Los tiempos combinados para un microprocesador, junto con la lógica de interrupción externa, determinan la rapidez con que el microprocesador responde a una petición de servicio de un dispositivo E/S.

El tiempo que transcurre entre la solicitud de una interrupción y el inicio de la ejecución de la subrutina de atención de la misma es el **tiempo de respuesta**, la suma de los tiempos de los pasos (1) y (2). La diferencia entre el tiempo total que ha sido interrumpido el microprocesador y el tiempo de ejecución de la subrutina de servicio propiamente dicha se denomina **overhead**[21]**.** Las estructuras de interrupción con un overhead bajo permiten un mayor throughput.

9.b. ESTRUCTURA DE INTERRUPCIONES DEL Z80

El microprocesador Z80 dispone de dos entradas para las interrupciones: NMI\ e INT\.

la pata NMI\ se utiliza para solicitar interrupciones no enmascarables. Permite sólo una interrupción de este tipo, que siempre es reconocida por la CPU. La entrada INT\ se utiliza para pedir interrupciones que se pueden enmascarar, y mediante un mecanismo de "vectoreo" que se explicará posteriormente, existe la posibilidad de tener 128 interrupciones distintas. Una solicitud de interrupción hecha a través de la pata INT\ solo será atendida si el *flip flop de habilitación de interrupciones interno del Z80, que llamaremos IFF1, está seteado.*

9.b.I. INTERRUPCIONES NO ENMASCARABLES

Cuando ocurre una interrupción no enmascarable (la pata NMI\ está baja cuando ocurre el último flanco creciente del último ciclo de reloj de una instrucción), es reconocida al final de la instrucción que se está ejecutando.

La CPU guarda en el stack el PC y salta a ejecutar en la posición 0066H (o sea, hace PC <-- 0 66H).

La rutina de servicio de una interrupción no enmascarable debe terminar con una instrucción RETN, que retira del stack el valor que tenía el PC al arribar la interrupción y lo pone en PC, lo que provoca que se siga ejecutando el programa principal.

El manejo de las interrupciones enmascarables durante una

[21]- Podríamos traducirlo como "sobrecarga".

atención a una NMI es un tanto sutil: en realidad existen dos flip flops internos al Z80 dedicados a las interrupciones, que llamaremos **IFF1** e **IFF2**. como ya dijimos, IFF1 está asociado con la habilitación y deshabilitación de las interrupciones enmascarables; se setea con la instrucción EI y se resetea con DI. IFF2 se utiliza para *almacenar temporalmente el estado de IFF1 durante el servicio a una NMI.* Además de guardar IIF1 en IIF2, se resetea IFF1, por lo cual no pueden ocurrir interrupciones enmascarables durante la atención de una NMI[22]. Otra acción de RETN, pues, será copiar el contenido de IFF2 en IFF1.

9.b.II. INTERRUPCIONES ENMASCARABLES

El Z80 admite tres modos de trabajar con sus interrupciones enmascarables. Para pasar de un modo a otro, existen 3 instrucciones: IM0, IM1 e IM2. Como veremos, el hardware para cada uno de los modos de funcionamiento es distinto, por lo cual una vez diseñado el sistema, el modo en que funcionan sus interrupciones debe ser siempre el mismo. Generalmente se ejecutará alguna de estas instrucciones en la rutina de inicialización del sistema. Por defecto, luego del reset la CPU queda en modo 0.

El microprocesador muestrea la pata INT\ en el flanco creciente del último ciclo de reloj de la ejecución de cada instrucción (no confundir con ciclo de máquina). Si INT\ = 0 y IFF1 = 1 (es decir, las interrupciones están habilitadas), se efectúa un ciclo de reconocimiento de interrupción enmascarable y se pasa a ejecutar la rutina de servicio de dicha interrupción. En la hoja de datos del Z80, podemos observar los diagramas de tiempos de dicho ciclo ("Interrupt Request/Acknowledge Cycle"). Vemos que *consiste en un ciclo M1 modificado, en el cual en lugar de bajar MREQ\, baja IORQ\.* Quiere decir que un dispositivo que desee conocer cuándo se está ejecutando un ciclo de reconocimiento de interrupciones enmascarables, deberá esperar por la condición IORQ\ = 0 AND M1\ = 0. Otra característica de este ciclo de máquina, es que la CPU le inserta automáticamente 2 ciclos de espera.

El flip flop de habilitación de interrupciones, IFF1, *es borrado al ejecutarse el ciclo de reconocimiento de la interrupción*. A diferencia de las NMI, aquí no se vuelve a la condición previa automáticamente, es decir, la única manera de que IFF1 = 1 nuevamente es ejecutar una instrucción EI.

En algunos casos, la última instrucción de la rutina de atención debe ser **RETI** para que la lógica externa de manejo de prioridades detecte el momento en que finaliza el servicio de la interrupción como veremos más adelante.

Si bien NMI e INT se muestrean en el mismo instante, de estar ambas activas tiene prioridad NMI.

[22] - Salvo que en la rutina de atención a la NMI se habiliten las interrupciones explícitamente.

9.b.II.1 Modo 0

Es el modo por defecto. Se incluyó en el Z80 por compatibilidad con el microprocesador 8080 de INTEL. En la bibliografía ([1], [6]) puede encontrarse una descripción detallada del mismo. No lo incluimos aquí por carecer de interés salvo en el caso en que se desea reutilizar periféricos pensados para el 8080.

9.b.II.2 Modo 1

Debe ejecutarse IM1 para que el Z80 ingrese en este modo.

En este modo, las acciones que tienen lugar si INT\ = 0 e IIF1 = 1 son análogas a las que tienen lugar cuando ocurre una NMI. La diferencia es que el PC se carga con el valor 0038H en lugar de la 0066H (valor para la NMI).

La gran ventaja de este modo es también su gran limitación: permite sólo una interrupción enmascarable, y como consecuencia, no es necesario hardware adicional para identificar al solicitante de la interrupción.

9.b.II.3 Modo 2

Es el más poderoso, y debe ejecutarse IM2 para que el Z80 quede en este modo. Permite 128 interrupciones provenientes de dispositivos externos.

El mecanismo utilizado es el llamado "vectoreo" de las interrupciones. En esencia, consiste en que el periférico que solicita la interrupción suministra además un número que lo identifica. Dicho número se utiliza como índice de una *tabla de vectores de interrupción* que contiene las direcciones de memoria donde comienzan las rutinas de atención a la interrupción de cada uno de los periféricos.

<u>la tabla de vectores de interrupción</u>

Esta tabla ocupa 256 bytes de memoria, y puede ubicarse en cualquier dirección de memoria: comienza en la dirección que resulta de multiplicar por $256_{10} = 100_{16}$ el contenido del registro **I**:

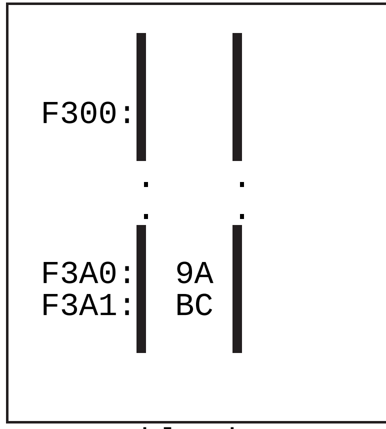


Contenido del
reg. I = F3H

En este caso, la tabla comenzará en la dirección F300H. Al solicitar la interrupción, el periférico debe poner en el bus de datos un número par $vvvvvvv0_2$, que representará el número de bytes que hay que desplazarse en la tabla de vectores de interrupción para llegar a la posición donde se encuentra almacenada la dirección de comienzo de la rutina de servicio de la interrupción.

Supongamos que $vvvvvvv0_2$ = A0H, que I = F3H y que la tabla es la indicada en el recuadro 11. La dirección de comienzo de la

rutina de servicio de la interrupción es pues BC9A H (recordar que el byte menos significativo va primero).



11 - Tabla de vectores de interrupción.

Queda por aclarar cómo llega al microprocesador el número de identificación del solicitante de la interrupción, vvvvvvv0. Si observamos el diagrama de tiempos del ciclo de reconocimiento de la interrupción, vemos que el periférico debe poner en el bus de datos un byte, que es leído al final del ciclo de reconocimiento de la interrupción. Para ello, el periférico espera la condición IORQ\ = 0 AND M1\ = 0.

Existen periféricos diseñados especialmente para el Z80 que implementan esta facilidad. En el capítulo siguiente estudiaremos uno de ellos, el Z80 PIO.

También se puede diseñar "a mano" un circuito que coloque el vector de interrupción en el bus de datos en el momento apropiado.

En la Fig. 23 vemos un caso en que se utiliza un buffer que pone en el bus de datos el valor F0H cuando detecta un ciclo de reconocimiento de una interrupción enmascarable. Uno de los inconvenientes que tiene este esquema es que no puede alterarse el vector utilizado.

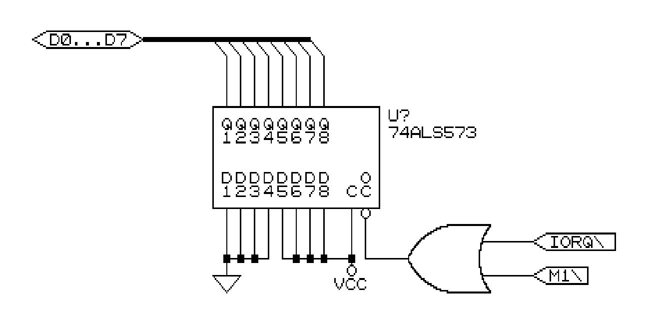


Fig. 23 Circuito que pone el vector de interrupción en el bus.

En la Fig. 24 vemos una solución más flexible: ahora se puede programar el valor que va a ponerse en el bus al detectarse un ciclo de reconocimiento de interrupción. En efecto, si el microprocesador desea que le coloquen el vector XX, alcanza con que al inicializar el sistema se ejecute la secuencia

```
LD  A, XX
OUT (80H), A
```

Debe notarse que cualquiera de estos métodos no son utilizables en caso que se desee tener más de una interrupción. Para ello es necesario agregar algo más de lógica, de modo de que el buffer solo ponga el número en el bus de datos si quien solicitó la interrupción es "su" periférico.

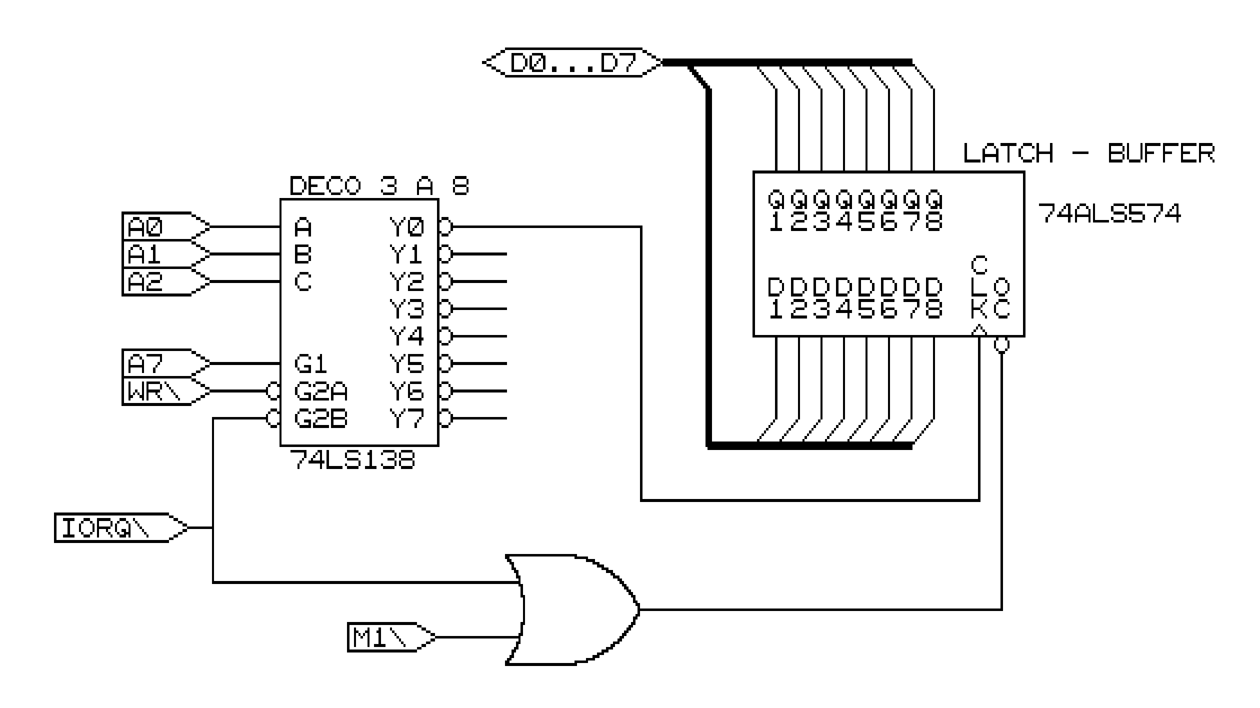


Fig. 24

9.c. EJEMPLO

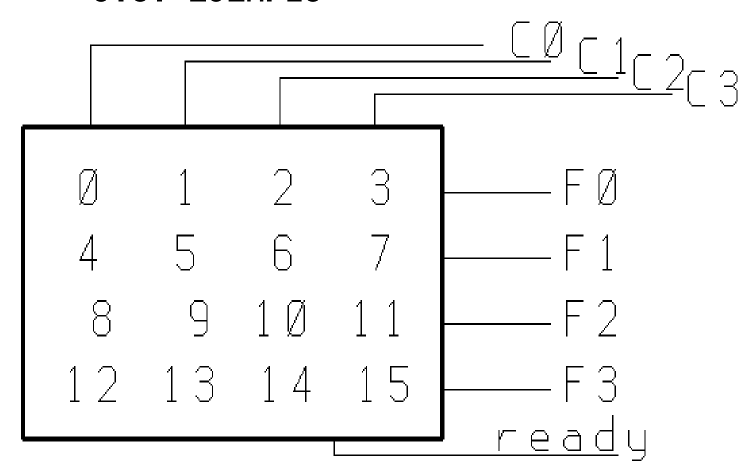


Fig. 25

El teclado para un dispositivo de entrada tiene 4 salidas **C0..C3** para indicar la columna correspondiente a la tecla apretada y 4 salidas **F0..F3** para indicar la fila, tal como se indica en la Fig. 25 . En reposo estas salidas están en nivel bajo y se activa una salida de cada grupo mientras se mantiene una tecla oprimida. Se dispone de una salida adicional **ready** que baja a cero durante 2 ciclos del reloj del sistema inmediatamente después que se oprime alguna tecla y se mantiene en 1 el resto del tiempo.

Se desea conectar este dispositivo a un microprocesador Z80 de manera que cada vez que se oprima una tecla se solicite una interrupción al Z80 a través de la entrada INT\ para leer a través del puerto de entrada 00H los valores que tomaron **C0..C3 F0..F3** cuando se oprimió la tecla.

En primer lugar, dibujaremos un diagrama detallado de la conexión del dispositivo. Supondremos que el teclado es el único dispositivo que solicita interrupciones y que la única dirección de

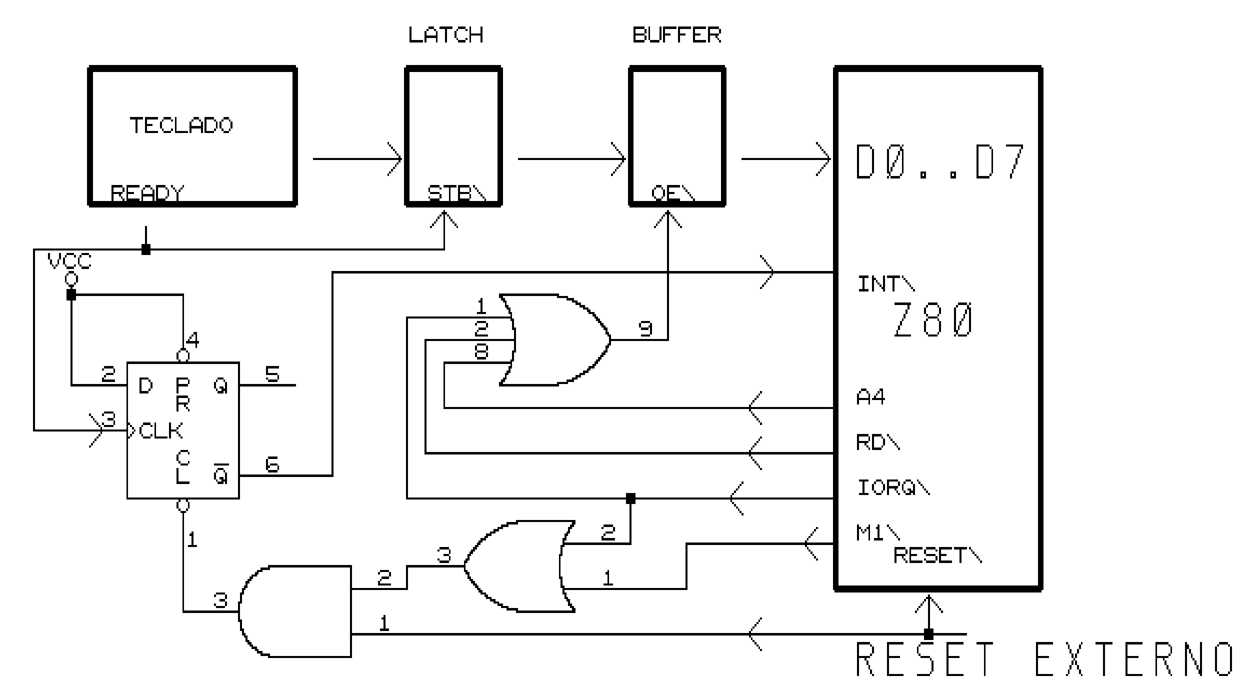


Fig. 26

E/S ocupada además de la 00H es la 10H.

Dado que es la única interrupción que existe, optamos por trabajar en modo 1.

En la Fig. 26 se ve una posible solución al problema. El pulso de READY provoca que los datos se almacenen en un latch, y a su vez provoca que la salida Q\ del flip flop se ponga a 0, provocando una interrupción. El flip flop es necesario pues debe garantizarse que la solicitud de interrupción (el 0 en INT\) se mantenga hasta que el microprocesador lo acepte. Cuando el microprocesador efectúa el ciclo de reconocimiento de la interrupción, bajan M1\ e IORQ\, por lo que se activa la pata CLEAR\ del flip flop, volviendo INT\ a 1. Como solo deben distinguirse las direcciones 00H y 10H de E/S, alcanza con utilizar A4 en la decodificación. Se garantiza que luego del reset no se solicitará una interrupción, ya que la línea RESET\ activa la entrada CLEAR\ del flip flop.

En cuanto al software, las instrucciones que deben ejecutarse en la inicialización del sistema para que el dispositivo funcione como se desea deben inicializar el SP, inicializar rl modo de interrupciones y habilitarlas:

```
LD SP, RAM_TOP+1     ;inicializo SP
IM1                  ; modo 1.
EI
```

Para finalizar el ejemplo, escribiremos la rutina de atención a la interrupción que lee del puerto 00H los valores de **C0..C3** y **F0..F3**, y escribe el resultado en la dirección de memoria 4000H.

```
ZONA ROM   EQU .....
```

```
     DEVOLUCION EQU 4000H

     ORG 0038H
          ; en el modo 1, el procesador comienza a ejecutar
          ; aquí el servicio de la interrupción.
     JP   AT_INTERRUP

     ORG ZONA_ROM
AT_INTERRUP:
     PUSH AF
     PUSH HL
     IN A, (00H)
     LD HL, DEVOLUCION
     LD (HL), A
     POP HL
     POP AF
     EI
     RETI
```

Algunas características de esta rutina son generales a todas las que atienden interrupciones.

- Como no se sabe en qué momento llega la interrupción, es indispensable "salvar el contexto". Esto quiere decir, que todos los registros que utiliza el procesador deben ser previamente guardados en el stack, y antes de terminar, restaurados. En el caso en que se pueda asegurar que no se utilizan en otro caso los bancos alternativos A’, F’, B’,...etc., pueden utilizarse aquí estos registros, con lo que se logra un cambio de contexto mucho más rápido que utilizando el stack.

- Debe ponerse especial cuidado en el manejo del stack. La regla a observar siempre es que *el número de PUSHs debe ser igual al número de POPs,para cualquiera de los posibles caminos que tome el programa*. En el caso del ejemplo, al no contener la rutina saltos condicionales, es trivial verificar que cumplimos con esta regla.

- Es razonable que siempre la última instrucción a ejecutar en una rutina de servicio de interrupción enmascarable sea EI. En efecto, con esto logramos dejar al procesador en las mismas condiciones que estaba inicialmente. Existen casos en que la ubicación de EI debe ser otra. Ejemplos extremos son por un lado, cuando se desea que dentro de una interrupción se atiendan otras interrupciones y por otro, cuando no queremos que se atiendan "nunca más" interrupciones.

- En el modo 1, es indiferente terminar con RET o con RETI.

9.d. "DAISY CHAIN": UN MECANISMO DE MANEJO DE PRIORIDADES

Cuando en un sistema existe más de un periférico que puede solicitar interrupciones, además del problema de identificar al solicitante (que en el caso del Z80 está resuelto si se trabaja en modo 2), aparece el problema de determinar prioridades entre los distintos periféricos. Este problema no se reduce al caso en que

vengan dos solicitudes de interrupción simultáneamente, sino que debe existir un mecanismo que permita que un dispositivo de alta prioridad pueda interrumpir a otro dispositivo de baja prioridad. Los periféricos diseñados especialmente para el Z80 traen incorporado un mecanismo para afrontar este problema, que se llama "daisy-chain". Estos periféricos dedican tres pines al control de interrupciones: INT\, IEI e IEO. Se unen todas las patas INT\ de los periféricos y se conectan a la pata de igual nombre del Z80. Utilizando una resistencia de pull-up (conectada a Vcc) se logra una estructura de OR cableado[23].

La entrada IEI (Interrupt Enable In) del dispositivo de mayor prioridad se conecta a Vcc., y la salida IEO (Interrupt Enable Out) a la entrada IEI del que le sigue en prioridad. La salida IEO se conecta al IEI del tercero, y así sucesivamente. *Para que un dispositivo pueda solicitar una interrupción, su entrada IEI debe estar alta. Además, inmediatamente antes de solicitar la interrupción baja IEO*. Con este esquema, es claro que el periférico de más alta prioridad siempre puede solicitar una interrupción, mientras que los demás solo podrán hacerlo si ningún periférico de mayor prioridad quiere solicitar una interrupción. A los efectos de determinar cuándo ha terminado el servicio de la interrupción, el periférico se queda "mirando" el bus de datos del Z80, a los efectos de detectar la instrucción RETI. Cuando la detecta, sube la línea IEO.

---

[23] - El OR cableado consiste en lo siguiente: las salidas INT\ cuando están inactivas en vez de poner un nivel alto, se ponen en alta impedancia, y el resultado final sigue siendo un nivel alto en la entrada INT\ del microprocesador, debido a la resistencia de pull-up. Si alguno de los periféricos desea solicitar una interrupción, baja su pata INT\ con lo que pone un 0 en la entrada INT\ del microprocesador sin dañar a las otras salidas INT\, que están en alta impedancia.

10. El Z80 PIO

10.a. INTRODUCCION

El Z80 PIO (Parallel I/O), es un chip de 40 pines diseñado especialmente para ser utilizado como interfaz entre el microprocesador Z80 y periféricos que acepten 8 bits en paralelo. Contiene dos puertos de 8 bits, que en la Fig. 27 se muestran como PA7..PA0 (Puerto A) y PB7..PB0 (Puerto B). Cada uno de ellos puede ser programado ya sea como puerto de entrada o de salida. Cada uno de ellos tiene, a su vez, dos líneas para handshaking entre el dispositivo de E/S y el PIO: RDY y STB\, que discutiremos más adelante.

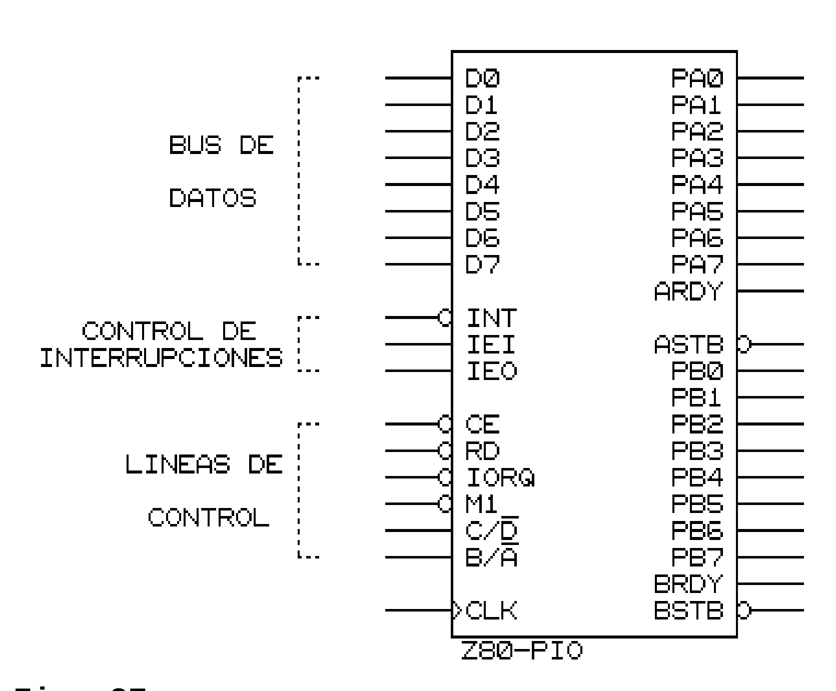


Fig. 27

Los datos se transfieren entre el PIO y la CPU por el bus de datos, D7..D0. Existen seis líneas que permiten al microprocesador controlar la operación del PIO.

* **B/A** (Port B/A select) selecciona uno de los puertos:
  = 0 ==> puerto A
  = 1 ==> puerto B.

* **C/D** (Control/Data select) indica al PIO el tipo de datos que se transfieren a través del bus de datos:
  = 1 ==> el bus de datos tiene una palabra de comandos para el PIO.
  = 0 ==> el bus de datos contiene datos.

* **CE** (Chip enable) un cero en esta pata indica al PIO que ha sido seleccionado para una operación de I/O.

* **M1** generalmente se conecta aquí la línea de igual nombre del Z80, que el PIO utiliza junto con IORQ\ para detectar ciclos de reconocimiento de interrupciones. Veremos que interviene también en el reset del PIO.

* **IORQ\ y RD** se conecta a las patas de igual nombre del Z80.

Como puede apreciarse en la Fig. 27 , existen también tres patas dedicadas al control de interrupciones, que funcionan de acuerdo al esquema de "daisy chain" descrito en el apartado 9.d.

10.b. ESTRUCTURA INTERNA: REGISTROS

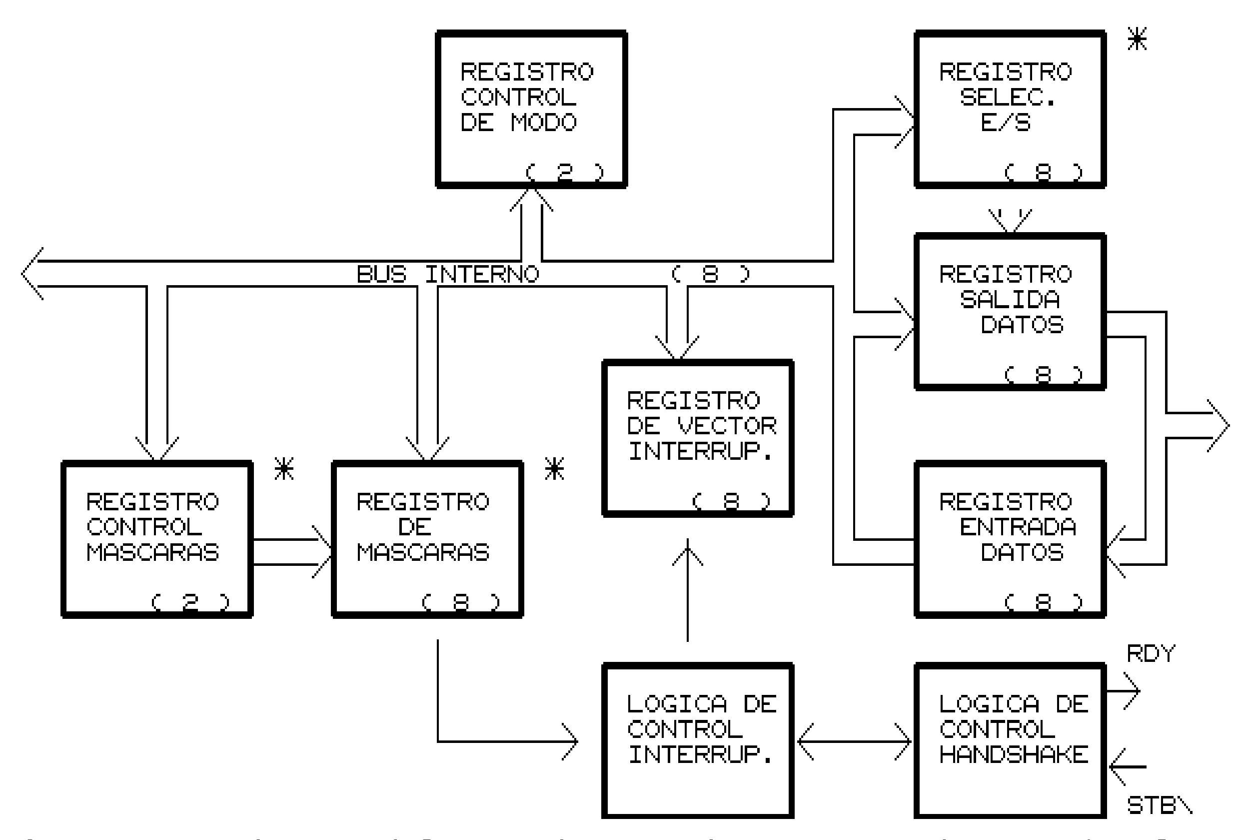


Fig. 28 - Registros del PIO. (Los registros marcados con * solo se utilizan en el modo de bit (modo 3) para permitir programar la generación de una interrupción con gran flexibilidad).

Internamente, cada puerto del PIO aparece como un conjunto de registros interconectados como se muestra en la Fig. 28 . Cada puerto contiene registros de entrada y de salida separados, lógica de control del "handshake", y ciertos registros de control, que se acceden, como vimos, poniendo la línea C/D\ alta. El registro de control principal tiene 2 bits, y se llama <u>registro de control de modo.</u> Existen 4 modos: salida, entrada, bidireccional y E/S de a bit.

El PIO está diseñado para ser utilizado con un Z80 cuyas interrupciones hayan sido programadas en el <u>modo 2 (vectorizadas)</u>, lo que implica que el PIO debe suministrar el vector de interrupción. A esos efectos, cada puerto tiene un <u>registro de vector de interrupción</u>, que puede ser cargado por la rutina de inicialización del sistema.

Como puede verse en la Fig. 28 , todos los restantes registros solo se utilizan en el modo 3 del PIO, E/S de a bit, y los explicaremos al tratar este modo de funcionamiento.

10.c. MODOS DE FUNCIONAMIENTO

El puerto A puede programarse en cualquiera de los 4 modos, en cambio el B puede estar solo en modo 0, 1 ó 3.

10.c.I. MODO 0: SALIDA

En este modo, el registro de salida de datos está activo y el de entrada de datos está inactivo (Ver Fig. 28). Los datos se escriben en el registro de salida del PIO ejecutando una instrucción OUT con la dirección de E/S apropiada. Los datos *pueden ser leídos nuevamente por el microprocesador, ejecutando una instrucción IN, con igual dirección*.

En cuanto a las líneas de protocolo[24], RDY sube cuando los datos se escriben en el puerto, indicando al periférico que los datos están disponibles. El dispositivo lee los datos bajando la pata STB\, lo cual resetea RDY y genera una interrupción al Z80 (si el PIO fue programado para generarla)

10.c.II. MODO 1: ENTRADA

El registro de entrada de datos está activo y el de salida inactivo. La secuencia de operaciones para entrar un dato al PIO es como sigue:

1. El dispositivo de entrada sensa RDY. Si está alto (lo que indica que el PIO está listo), el dispositivo pone los datos en el puerto y baja momentáneamente STB\.

2. Los datos se latchean en el registro de entrada del PIO. Esto resetea RDY y genera una interrupción (si así se programó).

3. El Z80 lee los datos del PIO, lo cual vuelve RDY a su estado activo (alto).

Para iniciar las operaciones en este modo, debe efectuarse una "lectura tonta" del puerto del microprocesador: consiste en leerlo y descartar los datos, pero logrando que RDY quede alta.

10.c.III. MODO 2: BIDIRECCIONAL

Es la superposición de los dos modos anteriores. Como se requieren cuatro líneas para handshake y dos vectores de interrupciones, sólo uno de los puertos (el fabricante eligió el puerto A) puede ser utilizado en este modo.

Las líneas de handshake del puerto A y su vector de interrupción se utiliza para implementar la salida de datos, y los de B para la entrada. Cuando A STB\ está baja, los datos del registro de salida del puerto A se ponen en las líneas de E/S del puerto. Cuando A STB\ está alta, pueden entrarse datos al registro de

---

[24] - Ver la hoja de datos del PIO para un diagrama de tiempo de las líneas de protocolo en cada uno de los modos.

entrada del puerto A, bajando B STB\. Las señales A RDY y B RDY pueden estar activas simultáneamente, indicando que existen datos disponibles para el periférico y que el PIO está listo para recibir datos de entrada.

Cuando A está en modo 2, el puerto B no dispone ni de el registro para el vector de interrupciones ni de líneas de hand-shake, por lo que solo puede ser programado en modo 3.

10.c.IV. PROGRAMACION DEL PIO EN MODOS 0, 1 Y 2

La programación de cada uno de los puertos del PIO en alguno de estos modos, requiere 2 palabras por puerto:

Una palabra de control de modo (Mode Control Word) que tiene la forma:

BIT 7	6	5	4	3	2	1	0
M1	M0	X	X	1	1	1	1

Mediante los bits identificados M1 y M0 se selecciona el modo.

Debe suministrarse también una vector de interrupción (Interrupt Vector Word), que consiste en un byte terminado en 0_2.

BIT 7	6	5	4	3	2	1	0
V7	V6	V5	V4	V3	V2	V1	0

Estas dos palabras también son necesarias para el modo 3, pero en ese caso, debe agregarse otras palabras de comando.

Notar que si se ha programado el puerto A en el modo 2, debe suministrarse también el vector de interrupción del puerto B.

10.c.V. MODO 3: E/S DE A BIT

En este modo, cada bit del puerto se define individualmente como entrada o salida. No utiliza las líneas de protocolo RDY y STB\. Se genera una interrupción si alguna de las entradas cambian, o si todas cambian (se ignoran los bits programados como salida a los efectos de la generación de la interrupción). Los requerimientos para generar la interrupción se definen durante la programación.

10.c.VI. PROGRAMACION DEL MODO 3

Para programar alguno de los puertos en este modo, se envía en primer lugar la palabra de control de modo, con sus dos bits más significativos en 1, e inmediatamente después,

- Una palabra de control del registro de E/S (I/O Register Control Word) que determina si cada uno de los bits son líneas de

entrada o salida

BIT	7	6	5	4	3	2	1	0
	D7	D6	D5	D4	D3	D2	D1	D0

Si D_i = 0, el bit i será salida, y en caso contrario, será entrada.

También debe enviarse una palabra de control de interrupción (Interrupt Control Word). En este modo, las interrupciones se generan como una función lógica de los niveles de las señales de entrada. Esta palabra de control define qué condición lógica y qué niveles deben darse para que se produzca una interrupción. Existen dos funciones lógicas disponibles: AND (Se genera una interrupción cuando todos los bits de entrada están en el estado activo) y OR (se genera una interrupción cuando alguno de los bits de entrada cambia al estado activo).

Permite especificar si luego de esta palabra se enviarán máscaras o no, mediante el bit 4.

BIT	7	6	5	4	3	2	1	0
	D7	D6	D5	D4	0	1	1	1

D7 = 0 : Interrupciones deshabilitadas.
   = 1 : Interrupciones habilitadas.

D6 = 0 : OR
   = 1 : AND

D5 = 0 : El nivel activo es bajo.
   = 1 : El nivel activo es alto.

D4 = 0 : No sigue palabra de máscaras
   = 1 : Sigue palabra de máscaras.

En caso que D4 = 1, el siguiente byte enviado al registro de control del puerto será interpretado como la palabra de control de máscaras (Mask Control Word):

BIT	7	6	5	4	3	2	1	0
	D7	D6	D5	D4	D3	D2	D1	D0

Este comando permite eliminar de la condición que genera la interrupción los bits que no se utilicen. Si D_i = 1, el bit i del puerto no participará en la condición de generación de interrupción.

Si se ha programado el PIO para que genere una interrupción, debe enviarse también el vector de interrupción (como en los otros modos). No se mencionó al principio, ya que en modo tres la palabra que debe seguir a la de control de modo es la de control del registro de E/S. Si inmediatamente después de la palabra de control de modo enviamos el vector de interrupción, este será interpretado por el PIO como la palabra de control del registro de E/S.

En efecto, como puede comprobar el lector, existen ciertas palabras de comando (por ejemplo, la de control de modo o la de control de interrupciones) que se caracterizan por tener algún patrón de bits en particular: por ejemplo, la de control de modo es la única que tiene cuatro "1" en los bits menos significativos. Esto posibilita que las enviemos en cualquier orden, ya que el PIO es capaz de reconocerlas. En cambio, las palabras de control del registro de E/S y la máscara de interrupciones son identificadas exclusivamente por la secuencia en que son enviadas. Así, el byte que siga a una palabra de control de modo que lleve al PIO a modo 3, será interpretada como control del registro de E/S.

10.c.VII. DESHABILITACION DE INTERRUPCIONES

Existe otra palabra de control que puede ser utilizada para habilitar o deshabilitar la generación de una interrupción por parte de un puerto. Puede ser utilizada en cualquiera de los modos del PIO.

BIT 7	6	5	4	3	2	1	0
I	X	X	X	0	0	1	1

Si I = 0, el puerto no generará interrupción. Si I = 1, si lo hará.

10.d. CONDICIONES INICIALES DEL PIO

El PIO se inicializa al encender el equipo o *llevando M1\ a cero mientras RD\ e IORQ\ están altas.* Esta última condición posibilita resetear el PIO sin apagar el equipo.

Las condiciones iniciales del PIO son:

1. Los flip flops de habilitación de interrupciones de los puertos, los registros de salida y los de máscara reseteados.
2. Modo 1 seleccionado (entrada).
3. Las líneas de E/S de los puertos en alta impedancia.
4. Las señales de handshake inactivas.
5. Los registros de vectores de interrupciones no reseteados.

10.e. EJEMPLO: INTERFAZ (PARALELO) CON UNA IMPRESORA

Mostramos aquí cómo interconectar el PIO a un microprocesador Z80, aprovechando para ilustrar su utilización con un ejemplo típico.

Supondremos que el sistema de este ejemplo solo puede ejecutar los programas que tiene en ROM, es decir, no tiene la posibilidad de "cargar" programas desde un periférico, como por ejemplo de una diskettera.

Consideremos una impresora que tiene una entrada de datos de 8 bits. Un flanco creciente en la entrada "dato válido", DAV, de la impresora provoca que esta lea los datos de entrada y los imprima. Luego de imprimir un carácter, la impresora emite un pulso (cero) en su salida "listo", RDY\.

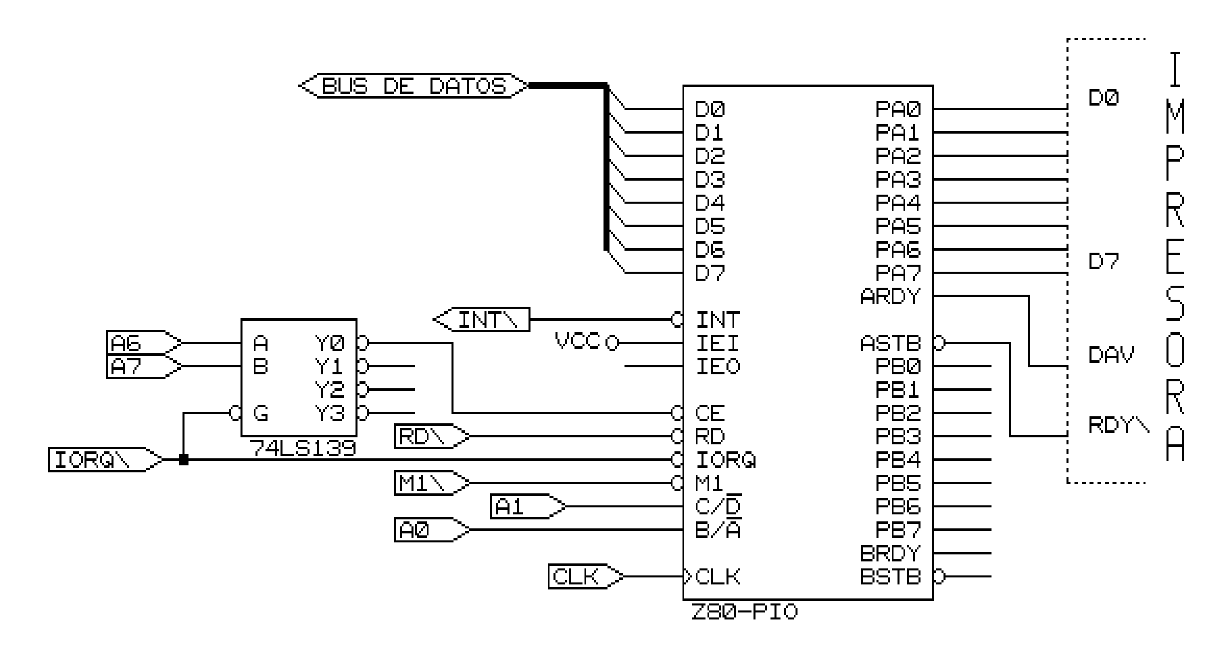


Fig. 29

En la Fig. 29 vemos un esquema de la conexión del PIO al Z80 y a la impresora. Describiremos en primer lugar la conexión del PIO al microprocesador. El bus de datos y las líneas de igual nombre se conectan directamente.

A los efectos de seleccionar el puerto A o B se utiliza la línea del bus de direcciones A0, y para seleccionar el registro de comandos o de datos, la línea A1.

A su vez, como el PIO se encuentra en la salida Y0 del decodificador, para acceder a él debe ponerse A7 = A6 = 0. Es decir que se verán los registros de acuerdo con la tabla Tabla V

Tabla V

DIR DE E/S	REGISTRO
00 H	DATOS, PUERTO A
01 H	DATOS, PUERTO B
02 H	COMANDOS, A
03 H	COMANDOS, B

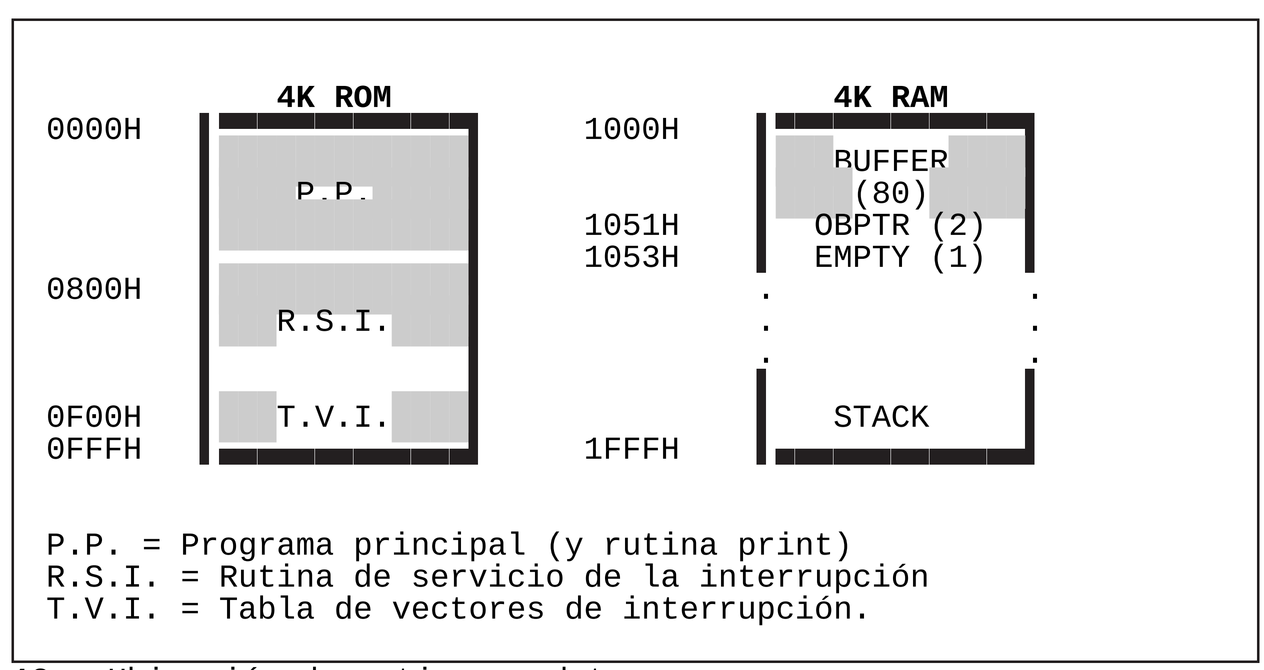
La presencia del decodificador solo se justifica si existen otras direcciones de E/S utilizadas. De lo contrario, bastaría con una compuerta OR.

Por el lado de la impre-

sora, de la descripción del problema se desprende inmediatamente la conexión a efectuar.

Consideremos que el sistema tiene ROM entre las direcciones 0H y 0FFFH, y RAM entre las direcciones 1000H y 2000H

Se presenta a continuación el software necesario para utilizar el sistema. Se eligió ubicar la tabla de vectores de interrupción en la dirección 0F00H (al final de la ROM), utilizar el vector 00H y ubicar la rutina de atención a la interrupción en la dirección 0800H (ver recuadro 12).



12 - Ubicación de rutinas y datos.

Durante la ejecución del programa principal, el microprocesador llena un buffer de memoria (RAM, obviamente) que comienza en la dirección OUTBUF, con caracteres ASCII que se imprimirán como una línea. Los dos últimos caracteres de la línea son el retorno de carro (0D H) y nueva línea (0A H). Luego que el microprocesador puso una línea completa de datos en el buffer, llama a la rutina PRINT. Esta inicializa el puntero del buffer, OBPTR, pone en cero la bandera EMPTY y habilita las interrupciones del PIO. Luego envía el primer byte a la impresora. Al aceptarlo, ésta genera una interrupción al Z80.

La rutina de servicio de la interrupción envía un byte y acomoda los punteros, esperando la siguiente interrupción. Si el byte enviado es el de nueva línea, deshabilita la generación de interrupciones por parte del PIO.

Entre tanto, el programa principal no enviará datos al buffer, ya que la bandera EMPTY valdrá 0. Esto significa que se está imprimiendo una línea, y por lo tanto, el buffer está siendo utilizado por la rutina de servicio de interrupción.

El software necesario tiene la forma:

```
; constantes referentes al PIO.

COMANDOS        EQU  02H       ; dir. de E/S del reg. de comandos
DATOS           EQU  00H       ; dir. de E/S del reg. de datos
MODO_SALIDA     EQU  00001111B ; comando: modo 0
NO_INTERR       EQU  00000011B ; comando: no interrumpir
SI_INTERR       EQU  10000011B ; comando: interrumpir

; constantes referentes al sistema de interrupciones del Z80.

VECT_INT   EQU  00H
INIT_TABLA EQU 0FH
LOW_INT    EQU  00H ; rutina de servicio a interrupción en la
HIGH_INT   EQU  08H ; dirección 0800H.


;programa principal

ORG 0000H

     IM2
     LD SP, 2000H    ; inicializo stack
     LD A, INIT_TABLA  ; inicializo tabla vect. ints.
     LD I, A

; programación del PIO

     LD A, MODO_SALIDA
     OUT (COMANDOS), A ; pio en modo 0
     LD A, VECT_INT
     OUT (COMANDOS), A ; programo vector de int. en PIO.
     LD A, NO_INTERR
     OUT (COMANDOS), A ; deshabilito interrupciones PIO
     EI
     .                 ; sigue el programa principal.
     .
     .

; subrutina para iniciar la impresión de un buffer:

PRINT:
     LD A, SI_INTERR
     OUT (COMANDOS), A ; habilito interrupciones PIO
     LD HL, OUTBUF  ; inicializo puntero al comienzo del buffer
     LD A, (HL)
     OUT (DATOS), A ; envío primer carácter de la línea.
     INC HL
     LD (OBPTR), HL ; salvo puntero.
     XOR A          ; reseteo bandera de buffer vacío
     LD (EMPTY), A  ;
     RET
```

```
ORG INIT_TABLA * 256 + VECT_INT
; la tabla de vectores de interrupción está en ROM, y por
; lo tanto la inicializo con la directiva DB

     DB LOW_INT
     DB HIGH_INT

; rutina de servicio de la interrupción:

ORG HIGH_INT * 256 + LOW_INT

     PUSH HL
     PUSH AF
     LD HL, (OBPTR) ; puntero al buffer
     LD A, (HL)     ; leo dato del buffer
     OUT (DATOS), A ; datos a la impresora
     INC HL         ; incremento puntero
     LD (OBPTR), HL ; salvo puntero
     CP 0AH         ; el carácter impreso es nueva línea?
     JR NZ, RESTOR  ; no, terminar rutina interrupción
     LD A, 0FFH     ; si, termina línea
     LD (EMPTY), A  ; buffer vacío
     LD A, NO_INTERR
     OUT (COMANDOS), A ; deshabilito interrupciones PIO
RESTOR:
     POP AF
     POP HL
     EI
     RETI

; reserva de memoria utilizada

ORG 1000 H

OUTBUF:   DS 80
OBPTR:    DS 2
EMPTY:    DS 1
     END
```

Para imprimir una línea de caracteres, el microprocesador llama a la subrutina PRINT, que imprime el primer carácter. Luego de esto, los caracteres se van imprimiendo a una velocidad determinada por la impresora. Cada vez que la impresora está lista, el microprocesador es interrumpido. Al imprimir el último carácter, se deshabilita la generación de interrupciones por parte del PIO, finalizando las transferencias.

En cuanto a la elección de las posiciones en memoria, la inicialización debe comenzar necesariamente en la posición 0000H de memoria (es lo primero que debe ejecutarse luego de un reset). La rutina print y la rutina de servicio de interrupción no tienen ningún requerimiento especial de ubicación, salvo que deben estar en ROM.

La tabla de vectores de interrupción puede implementarse tanto en ROM como en RAM. Si, por ejemplo, se hubiera elegido ubicarla en

la posición 1100H (es decir, en RAM), la tabla debe reescribirse cada vez que se "prende" el sistema, por lo cual la rutina de inicialización incluiría el siguiente trozo de código:

```
LD A, INIT_TABLA
LD I, A
LD A, LOW_INT
LD (INIT_TABLA * 256 + VECT_INT), A
LD A, HIGH_INT
LD (INIT_TABLA * 256 + VECT_INT + 1), A
```

Como en el ejemplo elegimos implementarla en ROM, alcanzó con grabar en las posiciones de memoria adecuada (en este caso, 0F00 y 0F01) la dirección de comienzo de la rutina de interrupción.

Bibliografía

[1] Barden Jr.,William, "The Z-80 Microcomputer Handbook, first edition", (Howard W. Sams & Co., 1978).

[2] Hill, Frederick J, Peterson, Gerarld R., "Digital Logical and Microprocessors", (John Wiley & Sons, 1984)

[3] Short, Kenneth L., "Microprocesadores y Lógica Programada", (Editorial Gustavo Gili, S.A., 1980).

[4] Short, Kenneth L., "Microprocessors and Programmed Logic, second edition", (Prentice-Hall International, 1987).

[5] Zacks, "Programación del Z80".

[6] "Zilog 1983/1984 Data Book".

INDICE